BAYESIAN WEIGHTED SUMMATION OF THE ABSOLUTE DIFFERENCE AND ITS IMPLICATIONS FOR ROBUST VISUAL NAVIGATION

By

ALEX JOHN SUHREN

Bachelor of Science in Mechanical Engineering

Oklahoma State University

Stillwater, OK

2013

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2016

BAYESIAN WEIGHTED SUMMATION OF THE ABSOLUTE DIFFERENCE AND ITS IMPLICATIONS FOR ROBUST VISUAL NAVIGATION

Thesis Approved:

Dr. Girish V. Chowdhary

Committee Chair and Thesis Advisor

Dr. Christopher J. Crick

Committee Chair

Dr. Jamey D. Jacob

Committee Chair

Acknowledgments

The author would like to acknowledge the following people:

Glen and Connie Suhren for their support both before and during the creation of this thesis.

Dr. Girish Chowdhary for the knowledge and guidance that he bestowed.

Dr. Christopher Crick for the use of his lab and robotic equipment.

Dr. Douglas Gaffin and Dr. Bradley Bradford for helping in the understanding of the biological aspect of this thesis, along with helping during the early stages of this thesis' creation.

Allan Axelrod, Ali Abdollahi, Rakshit Allamraju, and Sri Theja Vuppala for their help in proof reading and the organization of this thesis.

Lastly, the bees for there had work, and busy attitude. Without them this thesis would not bee possible.

Name: ALEX JOHN SUHREN

Date of Degree: JULY, 2016

Title of Study: BAYESIAN WEIGHTED SUMMATION OF THE ABSOLUTE DIFFERENCE AND ITS IMPLICATIONS FOR ROBUST VISUAL NAVIGATION

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: Visual navigation has a long history of taking inspiration from biology, owing to the wide range of applications facilitated by biological vision, such as feature detection and navigation. Of capital interest is the ability of small-brained organisms, such as bees and sub-Saharan ants, to navigate using dynamically changing landmarks and scenery. Biological research literature suggests that bees and ants can achieve this impressive feat by navigating toward scenes that are familiar. By contrast, the capability of computationally simple vision-based robotic methods, such as sum of the absolute difference (SAD), exhibit significantly degraded performance in changing environments. This disparity is addressed in this work, through a novel approach which determines regions within an image that are robust to environmental change by using the weighted sum of the absolute difference ($BWSAD$). By utilizing these regions, $BWSAD$ successfully navigates the paths in changing environments at a higher rate than the SAD baseline. 153 words

Contents

List of Tables

Table Page

List of Figures

Chapter 1

Introduction

1.1 Current state of the art in navigation

While there are many types of navigation techniques, principal among them includes simultaneous localization and mapping (SLAM) [6] [21] [22]. One of the most common methods of SLAM is done by utilizing a large array of sensors, which can include but are not limited to global position systems (GPS), inertial measurement units (IMU), light detection and ranging sensors (LiDARs), and cameras [33]. A common way that these algorithms work is by using the GPS to bound the drift created by integrating the IMU's acceleration data into position estimations. Then, a LiDAR and/or cameras are used to both detect obstacles, and are used to increase the accuracy of the localization data provided by the GPS and IMU [33]. Techniques that use such a vast array of sensors are often costly, physically cumbersome, and not applicable to small robots with limited computation bandwidth. To solve this, researchers have been working to limit the number of sensors to just a few cameras to obtain information about the surrounding environment, this is due to the dense amount of information that images can convey. Algorithms that rely on cameras as the primary external sensory units to navigate are known as vision-based navigation algorithms, and have proven to be both accurate, robust, and compact. To perform vision-based navigation, robots are typically fitted with cameras that are capable of producing high resolution images. These robots are then given navigation tasks that are considered remedial to most humans, but are done so to remove humans from unsafe environments, or to reduce the hazards caused by suboptimal human actions.

Vision-based navigation can be separated into two main categories, map-based [10] and mapless approaches [11] [13] [15]. Map-based approaches remember previously traversed regions of the world to provide a global understanding of where the robot is and the obstacles around it. The most common among the map-based navigation methods, is visual simultaneous localization and mapping (V-SLAM) [7] [8] [20], in which robots typically use Bayesian filtering methods, such as the particle

filter, to combine image feature data and odometry data. This allows the robot to obtain a 3-dimensional representation of image features. However, owing to the computational complexity of extracting image features and performing Bayesian filtering for non-Gaussian noise, these algorithms are computationally complex.

On the other hand, mapless navigation approaches are generally reactive in nature, meaning that they only seek to navigate using the current view of the robot without the aid of a global representation of the environment. There are two main methods of mapless navigation: optical flow based and appearance based navigation. In optical flow based navigation, optical flow from a camera is used to detect and then avoid obstacles [28] [29] [26]. In appearance based navigation, images or features are recorded from the environment during a training phase and this stored data is used to autonomously navigate through the environment. The work in this thesis falls under the domain of navigation known as appearance based navigation. Appearance based navigation algorithms are desirable in situations in which a robot is going to consistently travel the same area, such as a museum guiding robot [27], or situations in which the robot must back track to a home location, such as a Mars rover, autonomous construction equipment that must be put up after use, or search and rescue robots [25].

1.2 Related Work

It is believed by many biologist, that specific insects are capable of performing mapless visual navigation [3] [4]. One such insect is the bumblebee, an insect with poor visual capabilities and only 1 million neurons [19], as compared to 100 billion neurons in the human brain [12]. The bumblebee, in spite of these limitations, exhibits navigation behavior that is computationally efficient [3] [4]. Research shows that bumblebees are able to achieve this feat by navigating toward scenes that are familiar [9].

Biological researchers have also discovered that desert ants perform appearance based navigation [31] [30]. Most ants use pheromones to allow other ants to work together to reach a desired target location, however desert ants have not evolved this capability, since the pheromones evaporate in the hot and dry desert climates, rendering this navigation method useless. Instead it was discovered that desert ants use both a path integration method, similar to how IMU data is integrated to obtain position information, and a landmark based visual navigation technique that allows the ant to discern its location along the path. The ant is able to use integration data and landmarks to travel from one location to the visual catchment area of the next location [30]. Visual catchment

areas are local minimum of familiarity between the current view, and a view remembered from the ant's original path.

These biological discoveries have sparked an area of research known as visual homing navigation [2] [14]. In visual homing, the robot is given an image within an environment, and the robot can then travel to any 3D point in the environment. These kinds of algorithms works great when the original image is not occluded from the current view, however they are incapable of traveling to the home location when the original image is occluded from the current view. This means traveling in and out of rooms, is not a practical feat when using visual homing.

Another area of research similar to that of visual homing is visual memory navigation. In a sense, visual memory navigation can be thought of as an array of visual homing algorithms that have sequentially been strung together to form path. It is important to note that visual memory navigation is not a biologically inspired research field, however the similarities are quite remarkable. Visual memory navigation has two parts. The first is known as a the training run, in which a set of images or image features are recorded from a robot traveling along a path. The second is that of the autonomous navigation, outlined in Figure 3.1, in which the robot uses the images or features from the training run and compares them to the current image to determine control signals that the robot follows to retrace the original training path.

Visual memory navigation techniques can be separated into two main categories: model-based and view-based approaches. In model-based techniques, either image features or objects are extracted from the robot's current image, and are compared to features provided from the training run. Using these the robot is able to determine what control signals are required to reach the goal destination. These kind of methods can be formulated as a visual servoing problem [5]. In [25] a feature-based version of visual memory navigation is shown to have similar accuracy to that of navigation using a GPS. In [23] support vector machine (SVM) classifiers were used to both detect objects and localize along the training path. Image features are used to increase the robustness of image comparisons and to reduce the size of data storage. However, they are computationally expensive and recent work in V-SLAM has shown that impressive results can be obtained without feature detection [8].

View-based visual memory navigation techniques utilize the entirety of a given image to navigate instead of relying on image features to determine control signals. This work was pioneered most notably by Matsumoto et. al [17] in which he determined that a robot can follow a path when given images that were previously recorded while tele-operating along the path. This method used template matching to determine the familiarity between the two images. Later Matsumoto et. al [16]

found that by using an omni-directional camera the performance and reliability could be significantly improved. In [32] this technique was improved upon further by performing preprocessing on the training images, such as Fourier transforms and patch normalization to help increase the robustness to illumination variations and allow for navigation outdoors.

As with the aforementioned work in the field of visual memory navigation, the images taken by the robot are considered to be static and accurate representations of the environment. However, this is not often the case, as most environments are dynamic and subject to change. The key differentiation between this work and previous work is that in previous work the features or images taken from the training set are considered to be non-stationary and do not adapt. In this work the distribution of the scene is modeled as the path is retraced, thus negating the reliance on images taken from a single point in time. The algorithm proposed in this paper can be thought of an extension of Matsumoto et. al's [16] work and the template matching method known as the sum of the absolute difference, which forms the baseline for comparison in the experimentation.

1.3 Contributions

Herein, I present an analog to a simple visual memory navigation algorithm for robustly navigating a given path, provided initial training images along the path, which are subject to environmental variations. The contrast between my proposed approach and existing template-matching approaches [18] [16] is that we apply weights to the scenery using recursive Bayesian regression and weighted averaging techniques. Moreover, the recursive Bayesian regressions and weighted averaging yield an emergent behavior where, without requiring object recognition algorithms, robust visual anchors or landmarks are effectively identified based on the degree of variation in pixel values to the Bayesian-updated templates without requiring additional contextual information.

In summary the contributions of this paper are thus,

- The robot can use one training set to traverse two different lighting scenarios
- The robot can learn to increase the reliability of a simple visual memory navigation technique

Chapter 2

Problem Formulation

2.1 Robotic Navigation

The problem defined in this thesis is that of a robotic navigation problem. Simply put, a robotic navigation problem is one where a robotic agent is given the task of navigating from an initial location to a goal location in an environment with obstacles. More specifically, the type of navigation in this thesis is visual navigation. This is a navigation problem in which the primary sensor being used is a camera or cameras. The cameras are used to obtain images of the environment that are processed to come up with control signals to guide the robot through the environment and to the goal location. Specifically, for this problem, there is one camera that gives an omni-directional 360° view.

2.2 Vehicle model

Let s represent the state of the robot, as shown in Figure 2.1. In a planar world, this state is defined by the Cartesian coordinates (x, y, θ), where x and y are the Cartesian coordinates, and θ_I is the heading with respect to the inertial frame of the robot, such that, $s = SE(2) = \mathbb{R}^2 \times SO(2)$. The starting location of the robot is defined as $x = 0$, $y = 0$, and $\theta_I = 0$.

Let $R_B^I \in SO(2)$ represent the rotation matrix that transforms θ_I from the inertia frame to the body frame of the robot. Then,

$$\theta_B = R_B^I \theta_I \tag{2.1}$$

In this thesis, the robot is defined by its non-holonomic constraints.

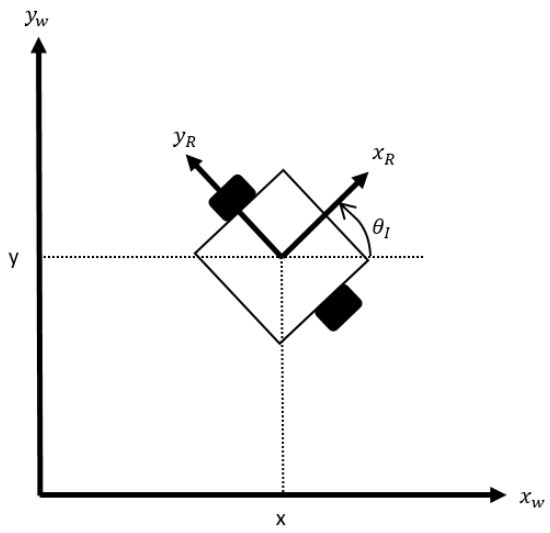


Figure 2.1: The robot's position and orientation are represented by x, y, θ_I in this picture. The global coordinate system is defined by x_w and y_w, and the robot's local coordinate system is defined by x_R and y_R.

2.3 Training Run

Before the robot can autonomously navigate a path, it must be given a set of images that were taken while the robot was tele-operated along the path. This tele-operation is defined as the training run and these images are defined as the training set.

The set of images, $\mathbf{I} = \{\mathbf{I}_0, \mathbf{I}_1, ..., \mathbf{I}_k, ..., \mathbf{I}_N\}$, represents the set of N training images taken during the training run, at a consistent interval. The set $T = \{T_0, T_1, ..., T_k, ..., T_N\}$ represents the set of N states that each image was taken at, where $\mathbf{I}_k$ was taken at state T_k.

2.4 Lighting and Environment Change

Now, consider that the environment the robot is traveling in is non-stationary in the sense that it is subject to visual regions of change over time due to changing lighting conditions and objects. Let $\mathbf{C}(s_a, t)$ represent the current image taken by the robot at state s_a and time t. We wish to create a set of probability distributions of the images as viewed from state s_a. This set of probability distributions is defined as $\mathbf{Q}(s_a)$, where there are $w \times h$ probability distributions that represent the probability of getting a certain pixel value from the camera at s_a. In a sense, $\mathbf{C}(s_a, t)$ represents a draw from $\mathbf{Q}(s_a)$.

2.5 Controller

Next, let e be the error in the heading defined as

$$e = \theta_B - \hat{\theta}_B \tag{2.2}$$

Lastly, define a proportional control law as,

$$\omega = pe$$

where ω is the control input to the robot denoted as angular velocity, and p is a proportional gain. The angular velocity command ω is passed to the existing motor controllers of the robot.

In short, this robotic navigation problem is the problem of determining the heading error e relative to the current location of the robot x and y, given a set of training images $\mathbf{I}_0$ obtained whilst the robot was tele-operated along a path.

Chapter 3

Solution

3.1 Baseline method

The navigation method proposed in this section is an extension of the view-based navigation method from [17]. In this method, a robot is teleoperated along a path and images are recorded at a steady frame-rate. Then the robot recapitulates the path by comparing the current image and the stored images to determine the desired heading that allows the robot to retrace the original path. This recapitulation is shown as a flow chart in Figure 3.1.

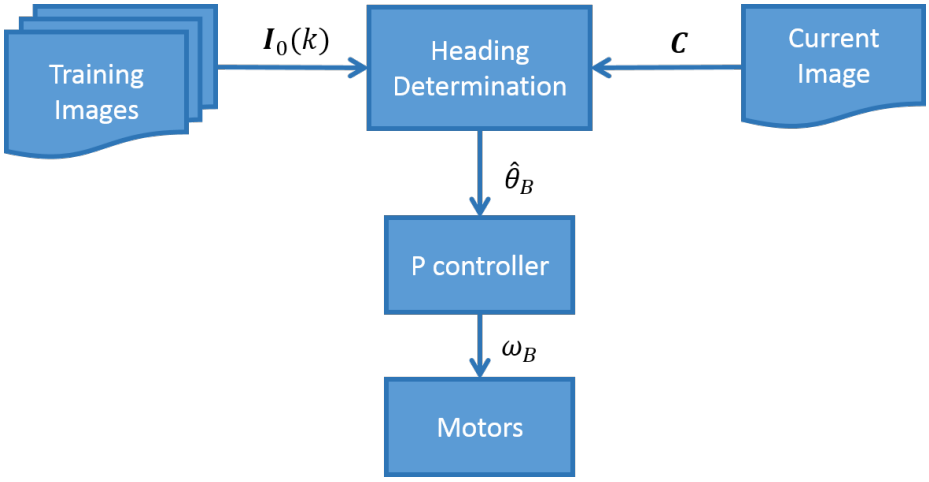


Figure 3.1: Flow chart of how visual memory navigation is performed

3.2 Determining the Heading

The original work done my Matsumoto used a front facing camera; however, in this paper a cata-dioptric omni-directional camera is used, as Matsumoto later found the images from such as device to be more reliable and robust [18].

Algorithm 1 Baseline SAD method

Input: $\mathbf{I}_0, \Delta\theta, \Delta k, p, v$
Output: $\hat{\theta}_B$
  $k = 1$
  Command robot to go forward at a velocity of v
  while robot is not at the goal location **do**
    $\mathbf{C}$ = CurrentImage
    k :=DetermineLocation($\mathbf{I_0}, \Delta\theta, \Delta k, \mathbf{C}, k$)
    ψ =DetermineHeading($\mathbf{I}_0, \mathbf{C}, k$)
    $\hat{\theta}_B = \psi\Delta\theta$
    $e = \hat{\theta}_B$
    $\omega = p * e$
    Command robot to turn at a rate of ω

This algorithm employs two comparison loops; one to determine location in terms of the images in the training set, k, and another to determine the heading $\hat{\theta}_B$. The location loop is defined in Algorithm 2, and it uses a large angular step size, $\Delta\theta$, to facilitate shorter computation times. The heading loop is defined in Algorithm 3 and it uses a fine angular step size of 1 to facilitate accuracy.

Algorithm 2 DetermineLocation

Input: $\mathbf{I}_0, \Delta\theta, \Delta k, \mathbf{C}, k$
Output: k
  for $\theta = 0 : \Delta\theta : 360$
    $\mathbf{R}$ =RotateImage($\mathbf{C}$, θ)
    for $i = 0 : \Delta k$
      $j = \theta/\Delta\theta$
      $\mathbf{C}_L[i, j] = SAD(\mathbf{R}, \mathbf{I}_0[k + i])$
  k := argmin$_i(\mathbf{C}_L) + k$

Algorithm 3 DetermineHeading

Input: $\mathbf{I}_0, \mathbf{C}, k$
Output: ψ
  for $\phi = 0 : 1 : 360$
    $\mathbf{R}$ =RotateImage($\mathbf{C}, \phi$)
    $\mathbf{C}_H[j] = SAD(\mathbf{R}, \mathbf{I}_0[k])$
  ψ = argmin$_\phi(\mathbf{C}_H)$

3.3 Baseline Comparison Metric

In [17] a circuit board was used to determine the similarity between two images, since computers were not yet capable of sufficiently fast computations. The specific method of comparison was not disclosed; however, it did use template matching. Consequently, the method of comparison selected for this paper is the sum of the absolute difference template matching metric or SAD for short. This metric was selected because of its computational simplicity and its ability to be used for the learning

process discussed later. Another popular template matching method that was considered, is the normalized cross-correlation coefficient (NCC). However, extension of the NCC template matching method into the proposed learning method is non-trivial, and is left up to future work.

For two images, **A** and **B**, with height and width, h and w respectively, SAD is defined below in equation 3.1.

$$SAD = \sum_{i=0}^{h} \sum_{j=0}^{w} |\mathbf{A}[i,j] - \mathbf{B}[i,j]| \tag{3.1}$$

3.4 Looking Ahead

When working with the previously mentioned baseline method, the robot was originally comparing the current image with all the training images. This facilitated a problem known as perceptual aliasing. Perceptual aliasing is the phenomenon that pertains to a robotic agent that incorrectly believes that the sensor readings from its location are from an incorrect location. The main issue here is that when the robot is traveling near the location of a training image, sometimes it can see the previous or next image as the correct image. When the robot is traveling in the middle of a straight line, this is not a problem, since the previous training image gives the same straight heading as the correct image, however when the robot turns the issue is exacerbated. For example, if the robot has to turn left, then the next image will tell it to turn left, however as it travels forward, the previous image looks more similar, so it may then start heading right. This causes the robot to navigate incorrectly. Experimentally, this issue appears to be resolved by only looking at images that are ahead of the current image.

3.5 Learning algorithm

In order to robustify the baseline algorithm, a simple learning algorithm is utilized such that the robot does not rely solely on one training set of images. Instead the robot learns what parts of the images within the training set are more reliable and robust. To do this, a collection of Gaussian distributions are calculated that best describes the true distribution of each pixel within all the training images. From here the variance of each pixel's Gaussian can give insight into what parts of the image have varied too much to be reliable and robust.

These distributions are learned through prior-posterior chaining, a Bayesian statistical method of estimating a true distribution by combining sampled distributions into a single estimated distri-

bution. In prior-posterior chaining, the prior, the probability of the current hypothesis, is multiplied by the likelihood, the probability of the observed data given the current hypothesis, to obtain a distribution proportional to the posterior distribution. This relationship is known as Bayes rule and is defined below in equation 3.2.

$$P(H|O) \propto P(O|H)P(H) \tag{3.2}$$

To create the chaining part of this process, the posterior distribution becomes the new prior distribution. Then when more data comes in, the likelihood is formed and multiplied by the new prior. A new posterior distribution is obtained and the whole process is repeated creating a chain that allows the posterior to be continuously updated based upon incoming data.

3.6 Distribution selection

3.6.1 Likelihood

The likelihood is represented by a Gaussian distribution, with unknown mean and unknown variance, that represents a single pixel of the current view of the robot where the mean, μ, is the true average pixel value between 0 and 255, and λ is the true precision. This Gaussian distribution is shown in equation (3.3) [24], where the variables, x and $\bar{x}$ represent the obtained pixel value and the average of all pixel values taken at that time step and pixel location.

$$p(D|\mu,\lambda) = \frac{1}{(2\pi)^{n/2}}\lambda^{n/2} exp\left(-\frac{\lambda}{2}\left[n(\mu-\bar{x})^2 + \sum_{i=1}^{n}(x_i-\bar{x})^2\right]\right) \tag{3.3}$$

3.6.2 Conjugate prior

An important aspect in prior-posterior chaining is that the prior and posterior must be the same kind of distribution. To do this, a prior must be selected such that multiplying by the likelihood, results in a posterior distribution that is of the same kind as the prior. A prior distribution that meets this qualification is known as a conjugate prior. For this case the likelihood is a Gaussian distribution that has unknown mean and unknown variance. The conjugate prior to this kind of distribution is the normal-Gamma distribution shown in equation (3.4) and (3.5), [24]. The normal-Gamma prior is a distribution of the mean and precision of a Gaussian distribution and it has parameters μ_0, κ_0, α_0, and β_0, where μ_0 is the mean, $\kappa_0 > 0$ is a gain term used to determine how much emphasis should be placed on the original prior, and α_0 and β_0 are terms used to calculate the precision.

$$NG(\mu, \lambda|\mu_0, \kappa_0, \alpha_0, \beta_0) = \frac{1}{Z_{NG}}\lambda^{\alpha_0 - \frac{1}{2}} exp(-\frac{\lambda}{2}[\kappa_0(\mu - \mu_0)^2 + 2\beta_0]) \tag{3.4}$$

$$Z_{NG}(\mu_0, \kappa_0, \alpha_0, \beta_0) = \frac{\Gamma(\alpha_0)}{\beta_0^{\alpha_0}} \left(\frac{\kappa_0}{2\pi}\right)^{\frac{1}{2}} \tag{3.5}$$

where μ and λ are the mean and precision of the training set, respectively.

3.6.3 Posterior

The posterior, normal-Gamma distribution is created by multiplying the normal-Gamma prior and the Gaussian likelihood. The calculation of the posterior's parameters are defined in Murphy's paper [24] over conjugate Bayesian analysis of the Gaussian distribution, and they are shown below in equations (3.6:3.9).

$$\mu_n = \frac{\kappa_{n-1}\mu_{n-1} + n\bar{x}}{\kappa_{n-1} + n} \tag{3.6}$$

$$\kappa_n = \kappa_{n-1} + n \tag{3.7}$$

$$\alpha_n = \alpha_{n-1} + n/2 \tag{3.8}$$

$$\beta_n = \beta_{n-1} + \frac{1}{2}\sum_{i=1}^{n}(x_i - \bar{x})^2 + \frac{\kappa_{n-1}n(\bar{x} - \mu_{n-1})^2}{2(\kappa_{n-1} + n)} \tag{3.9}$$

It is important to note that the number of data points, n, is 1 for all cases since there is only one image at each time step. Do to this, $x = \{x_1\}$, and the average pixel value, $\bar{x}$ is equivalent to x_1. Thus,

$$x_1 - \bar{x} = 0 \tag{3.10}$$

and

$$\frac{1}{2}\sum_{i=1}^{n}(x_i - \bar{x})^2 = 0 \tag{3.11}$$

Thus equations (3.6:3.9) can be simplified to,

$$\mu_n = \frac{\kappa_{n-1}\mu_0 + x}{\kappa_{n-1} + 1} \tag{3.12}$$

$$\kappa_n = \kappa_{n-1} + 1 \tag{3.13}$$

$$\alpha_n = \alpha_{n-1} + 1/2 \tag{3.14}$$

$$\beta_n = \beta_{n-1} + \frac{\kappa_{n-1}(x - \mu_{n-1})^2}{2(\kappa_{n-1} + 1)} \tag{3.15}$$

3.7 Autonomous Run and Learning Implementation

Once the robot has a training set, it is can attempt to retrace the path. To do this, the robot compares images within the training set to the images it is currently seeing through the camera. An overview of this method is shown in Figure 3.2. Notice how the weights are determined from the previous training images, and are then used to

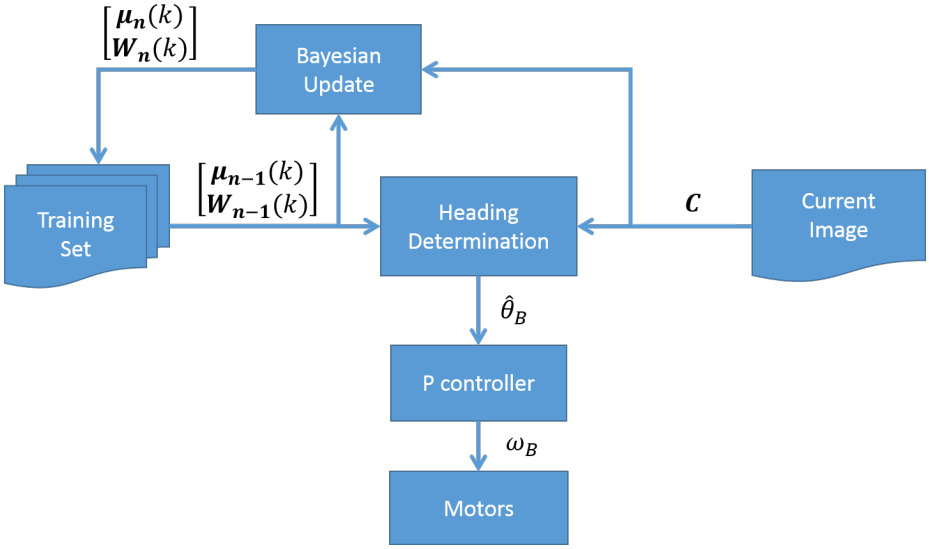


Figure 3.2: Outline of how the new Bayesian Weighting method from this thesis works.

Determining the heading is done in the same way as the baseline method; however, the comparison method used, shown in equation 3.16, is a weighted version of the SAD metric, denoted as the Bayesian weighted sum of the absolute difference ($BWSAD$).

$$BWSAD = \sum_{i=0}^{h}\sum_{j=0}^{w} \mathbf{W}[i,j]|\mathbf{A}[i,j] - \mathbf{B}[i,j]| \tag{3.16}$$

where $\mathbf{A}$ and $\mathbf{B}$ are images with the same height and width, h and w, and $\mathbf{W}$ is a binary weighting matrix, meaning that $\mathbf{W}[i,j]\exists\{0,1\}\forall i,j$ as determined by Algorithm 4.

The images are then compared in the same structure that Algorithm 1 has; however, after determining the proper index in the training set, and the heading, an update of the current posterior distribution is performed and can be seen in Algorithm 4. The end result is Algorithm 5 which shows how this method of navigation works as a whole.

Algorithm 4 Update

Input: $\mathbf{C}, k, \epsilon, \mu_0, \kappa_0, \alpha_0, \beta_0, \mathbf{W}$
Output: $\mu_n, \kappa_n, \alpha_n, \beta_n, \mathbf{W}$
  for $i = 0$: height of $\mathbf{C}$
    for $j = 0$: width of $\mathbf{C}$
      $\mu_n[i,j,k] = (\kappa_0[i,j,k]\mu_0[i,j,k] + \mathbf{C}[i,j])/(\kappa_0[i,j,k]+1)$
      $\kappa_n[i,j,k] = \kappa_0[i,j,k] + 1$
      $\alpha_n[i,j,k] = \alpha_0[i,j,k] + 1/2$
      $\beta_n[i,j,k] = \beta_0[i,j,k] + \kappa_0[i,j,k](\mathbf{C}[i,j] - \mu_0[i,j,k])^2/(2(\kappa_\mathbf{0}[i,j,k]+1))$
      $\lambda_0 = \alpha_n./\beta_n$
      if $\lambda < \max(\lambda)/\epsilon$ **then**
        $\mathbf{W}[i,j,k] = 1$
      else
        $\mathbf{W}[i,j,k] = 0$

Algorithm 5 BWSAD Navigation

Input: $\mathbf{I}_0, \Delta\theta, \Delta k, \epsilon, \omega, v$
Output: h
  $k = 1$
  Command robot to go forward at a velocity of v
  while robot is not at the goal location **do**
    $\mathbf{C}$ = CurrentImage()
    k :=DetermineLocation($\mu_0, \Delta\theta, \Delta k, \mathbf{C}, k, \mathbf{W}$)
    ψ =DetermineHeading($\mu_0, \mathbf{C}, k, \mathbf{W}$)
    $\mathbf{B}$ =RotateImage($\mathbf{C}, \psi$)
    $[\mu_0, \kappa_0, \alpha_0, \beta_0, \mathbf{W}]$ =Update($\mathbf{C}, k, \epsilon, \mu_0, \kappa_0, \alpha_0, \beta_0, \mathbf{W}$)
    $e = \hat{\theta}_B$
    $\omega = p * e$
    Command robot to turn at a rate of ω
  Stop the robot

Algorithm 6 DetermineLocation

Input: $\mu_0, \Delta\theta, \Delta k, \mathbf{C}, k, \mathbf{W}, \mu_\mathbf{n}$
Output: k
  for $\theta = 0 : \Delta\theta : 360$
    $\mathbf{R}$ =RotateImage($\mathbf{C}$, θ)
    for $i = 0 : \Delta k$
      $j = \theta / \Delta\theta$
      $\mathbf{C}_L[i, j] = BWSAD(\mathbf{R}, \mu_0[k + i], \mathbf{W})$
  $k := \text{argmin}_i(\mathbf{C}_L) + k$

Algorithm 7 DetermineHeading

Input: $\mu_0, \mathbf{C}, k, \mathbf{W}$
Output: ψ
  for $\phi = 0 : 1 : 360$
    $\mathbf{R}$ =RotateImage($\mathbf{C}, \phi$)
    $\mathbf{C}_H[j] = BWSAD(\mathbf{R}, \mu_0[k], \mathbf{W})$
  $\psi = \text{argmin}_\phi(\mathbf{C}_H)$

3.8 Controller

A proportional control law was used and is defined as,

$$\omega = p * e \tag{3.17}$$

where, ω is the commanded angular velocity of the robot, p is the proportional controller's gain, and e is the heading error.

Chapter 4

Results

Since the principle contribution of this algorithm is the increased robustness of a simple visual navigation algorithm, Monte Carlo runs are desired to obtain accuracy and robustness data. Due to the innate repeatability of simulations, when compared to real world testing, a simulation was setup to test the theoretical feasibility of the proposed algorithm.

4.1 Simulation Setup

The robotic operating system (ROS) was used to communicate with the simulation environment and control the simulated robot. As a middle-ware architecture to interface between commonly used robots and robotic algorithms, ROS is widely used, well supported, and very portable between different robotic architectures. This made ROS an obvious choice to enable ease of use along with reliable results. The robot that was simulated is the Turtlebot 2, a robot that is widely used due to its simple design and innate compatibility with ROS. The model used to simulate the Turtlebot 2 is shown in Figure 4.1. The simulation software known as Gazebo was picked due to the close relationship that Gazebo and ROS have. As Gazebo initially came as a package in ROS, it was a natural choice to consider. Moreover, Gazebo comes with a simulation of the Turtlebot 2 with all the same ROS capabilities of a real world Turtlebot making the real world extensions simple.

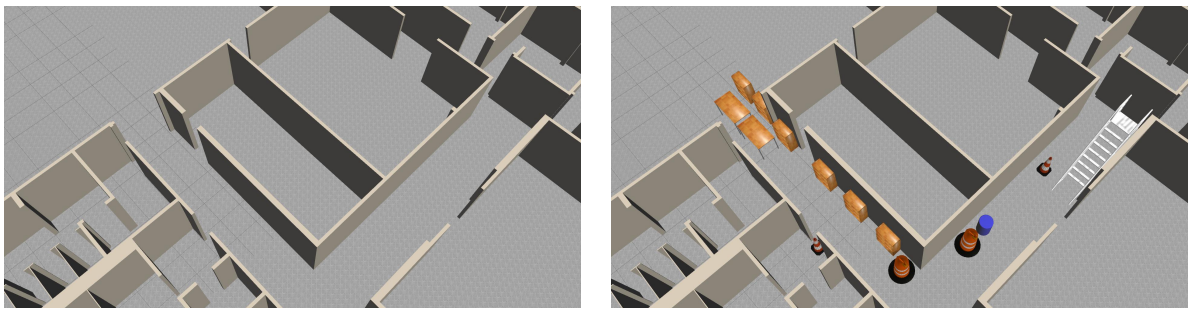
The world file used for this simulation is the Willow Garage file shown in Figure 4.2(a), which is a 3D model of the company Willow Garage's office space. This world file was picked because of its similarity to a typical office building, and its similar looking corridors, which pose as a difficulty to navigation schemes, do to the low number of features. Since the SAD method navigates using scene familiarity, objects were added into the scene to make it less familiar to the training images and consequently more difficult for the robot to autonomously traverse the path. This was also done to show that the proposed learning algorithm can get past these hurdles; this cluttered simulation

Figure 4.1: Example image of a Turtlebot 2, a mobile robot with a 3D Microsoft Kinect sensor, a bump sensor, and a net book. This was the robot that was simulated during testing due to its popularity, simplicity, and compatibility with ROS

environment is shown in Figure 4.2(b).

4.2 Camera Simulation

Along with the Turtlebot, a catadioptric omni-directional camera was simulated. This omni-directional camera was created by taking 6 cameras, each facing in 6 orthogonal directions away from each other, i.e, $(x, y, z, -x, -y, -z)$, and then applying the image transform below in equations 4.1, 4.2, and 4.3 to convert the images from a rectangular image frame to a spherical image frame.



(a) Training Run Simulation

(b) Autonomous Run Simulation

Figure 4.2: This figure is designed to show what steps were taken to confuse the baseline algorithm, and to show that the learning algorithm is able to determine what parts of the scene have changed and are thus not reliable or robust. The top picture (a) shows the simulated office space that was used to train the robot. The bottom picture (b) shows the same office space after various objects have been added to the scene in order to confuse the robot while trying to navigate the environment.

$$x = cos(\theta)cos(\phi) \tag{4.1}$$

$$y = sin(\theta)cos(\phi) \tag{4.2}$$

$$z = sin(\phi) \tag{4.3}$$

where θ and ϕ are the azimuth and elevation angles used to fully define a unit sphere.

Then the spherical image with width, w, and height, h, and pixels at (i, j) is converted to a catadioptric omni-directional camera image with pixels at (x, y) by applying the following transformation.

$$\alpha = atan2(-i, j) \tag{4.4}$$

$$r = \sqrt{i^2 + j^2} \tag{4.5}$$

$$x = \alpha * \frac{w}{2\pi} \tag{4.6}$$

$$y = h * (\frac{r - r_i/2}{(R_o - r_i)/2})^2 \tag{4.7}$$

where r and R_0 are the inner and outer radius respectively, as shown in Figure 4.3.

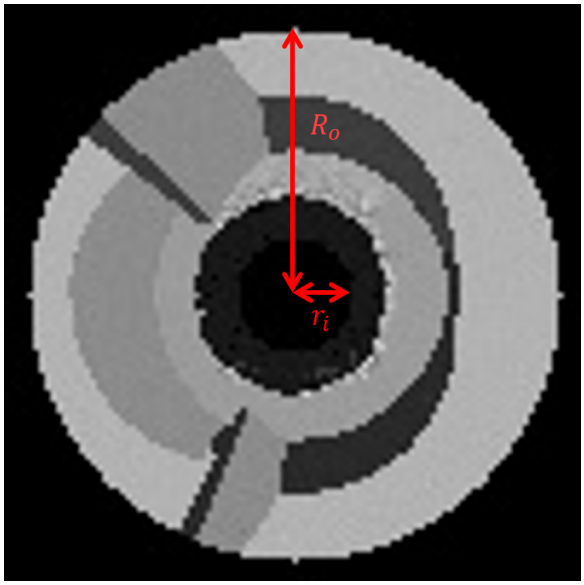


Figure 4.3: This picture shows what the variables R_0 and r_i represent. Note that r_i is selected to occlude the Turtlebot, as it does not rotate with respect to the camera

4.3 Simulation Runs

All simulations are done in real-time including the physics, visual rendering, and the aforementioned camera transform. Also, the environment and the Turtlebot used are on a 1 to 1 scaling with respect to Gazebo.

4.3.1 Training Run

The Turtlebot was first tele-operated along a basic corridor shown in Figure 4.4, while both the omni-directional images and location of the Turtlebot were recorded at $2hz$. This set of images is known as the training set, $\mathbf{I}_0$ An example of a few of the images recorded during the training run are shown in Figure 4.5.

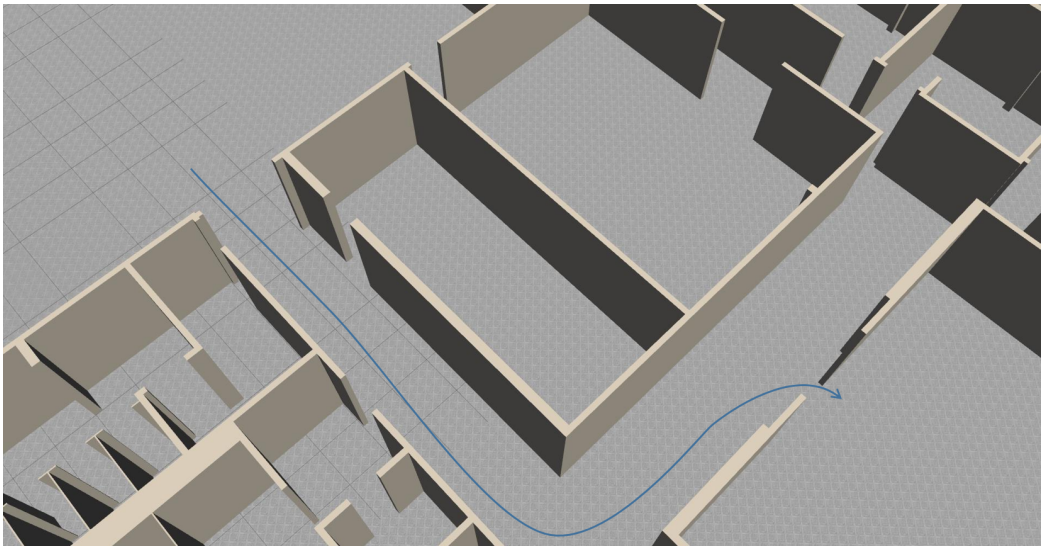
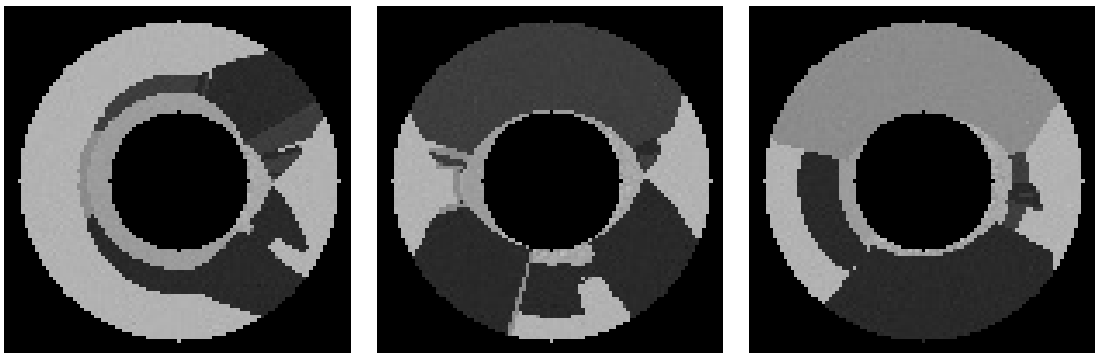


Figure 4.4: This figure shows the Gazebo world that was used to make the training path, along with path that the robot was tele-operated along. This path starts on the left and ends at the right.

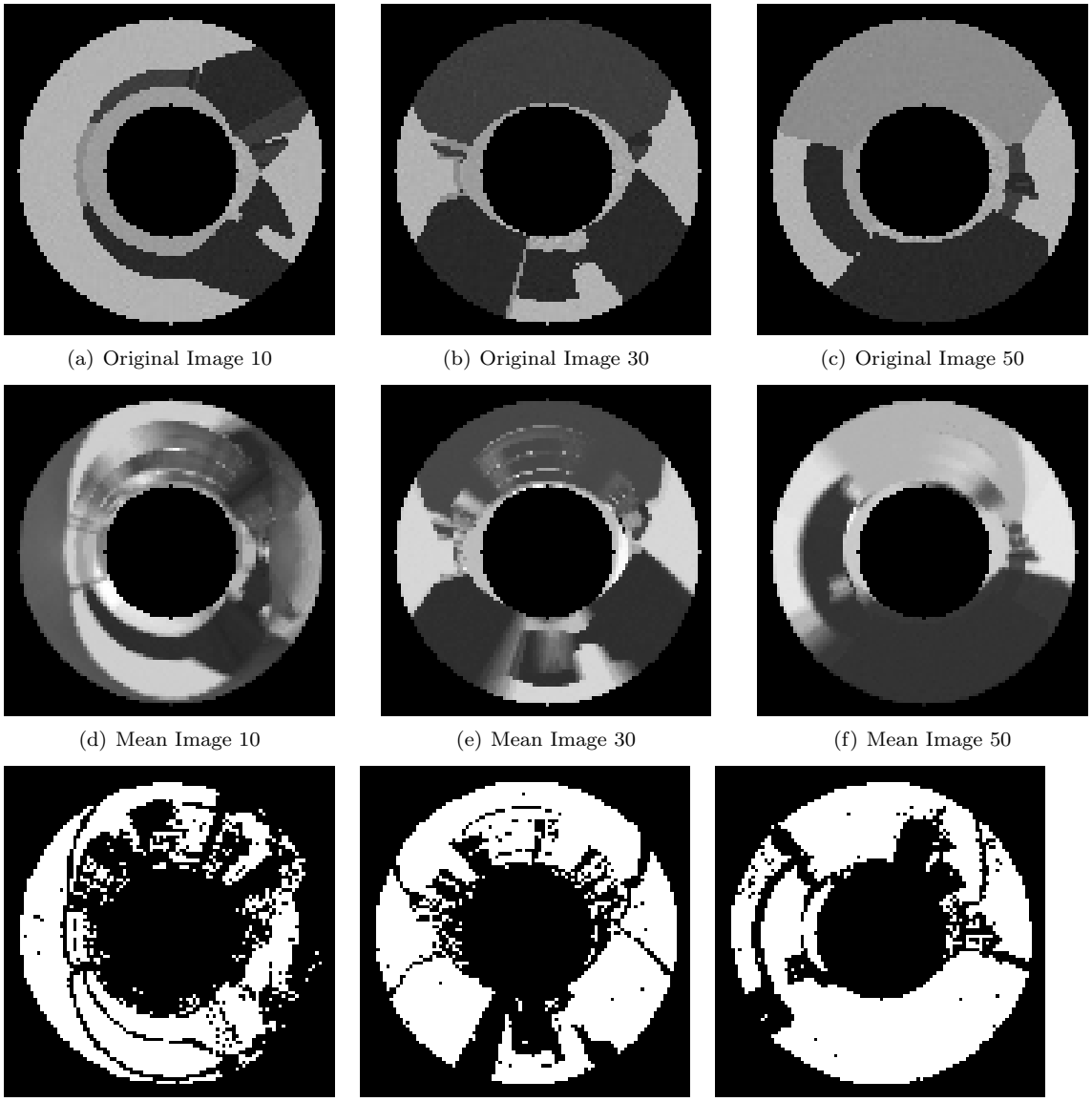


(a) 10th Training Image (b) 30th Training Image (c) 50th Training Image

Figure 4.5: This figure shows three examples of images that were taken during the training run, specifically 10, 30, and 50.

4.3.2 Autonomous Run

Once the Turtlebot has a training set, it then recapitulates the path 2000 times, where the first 1000 runs use the baseline SAD method without weighting. As the robot does the first 1000 runs, every training image $\mathbf{I}_0[k]$ in the training set, $\mathbf{I}_0$, is given a matrix of means, $\mu_0[k]$, and a matrix



(a) Original Image 10 (b) Original Image 30 (c) Original Image 50

(d) Mean Image 10 (e) Mean Image 30 (f) Mean Image 50

(g) Weighting Image 10 (h) Weighting Image 30 (i) Weighting Image 50

Figure 4.6: This figure shows the mean (μ_0), weighting (ω), and original images of the learning algorithm after 1000 runs. The top images show the original $10th$, $30th$, and $50th$ training images in order from left to right without any clutter in the scene. The middle images show the mean of the $10th$, $30th$, and $50th$ training images in order from left to right after the learning of roughly 1000 runs was achieved. The bottom three show the weighting applied to the comparison metric where white is 1 and black is 0.

of weights, $\mathbf{W}[k]$ defined in Algorithm 4. During each autonomous run, any current images that are determined to be at the same location as the training image $\mathbf{I}_0[k]$, are integrated into $\mu_0[k]$ and $\mathbf{W}[k]$ using Algorithm 4. However, the weights are not used during the first 1000 runs, they are saved for future use during the last 1000 runs.

To understand what the mean and weighting terms look like, Figure 4.6 was created, which shows these two terms along with the original training image for three of the training images. Notice how the three images that show the mean have blurred regions, which correspond to the locations of the bookcases and other objects added to Figure 4.2(b), and they are shown as black in the weighting. This is because the blurred regions of the image represent pixels that vary from frame to frame, thus the weighting puts no emphasis on these segments of the image.

At run 1001 the robot switches to apply the weights, $\mathbf{W}$, obtained only from the successful previous runs. These 1000 runs use the $BWSAD$ navigation method and the robot no longer updates the posterior. This provides consistency for the last 1000 Monte Carlo runs, which have learned off the successful runs that used the SAD navigation method.

When running this simulation it is important to have success and failure criteria to accurately determine how often the methods have succeeded. The failure of the robot to follow the path is automatically determined if the robot remains at the same area for 10 secs or the robot traverses 1.5 Gazebo distance units, meters, away from the training path shown in Figure 4.4. The robots success is determined by the robot's ability to get within 1 Gazebo distance unit, meter, of the goal location.

4.4 Simulation Results

The parameters of the baseline and learning method are tabulated below in Tables 4.1 and 4.2, where $\Delta\theta$ is the location of the rotation comparison, used to determine what image the robot is at, Δk is the number of images that the robot looks though when finding the best training image, ϵ is the gain used to determine what parts of the image are more robust, and p is the proportional gain used to provide a proportional controller for the robot. These 4 variables are detailed in Algorithms 1 and 5.

Table 4.1: Parameters used for SAD

	$\Delta\theta$	Δk	p
Values	12	4	6

The tests described previously were performed three times, with the objects moving to different

Table 4.2: Parameters used for $BWSAD$

	$\Delta\theta$	Δk	ϵ	p
Values	12	4	150	6

locations. For the $1st$ simulation the autonomous run was performed in the environment shown in Figure 4.7. The $2nd$ and $3rd$ simulations were run in the environment shown in Figure 4.2(b). The results are tabulated in Table 4.3.

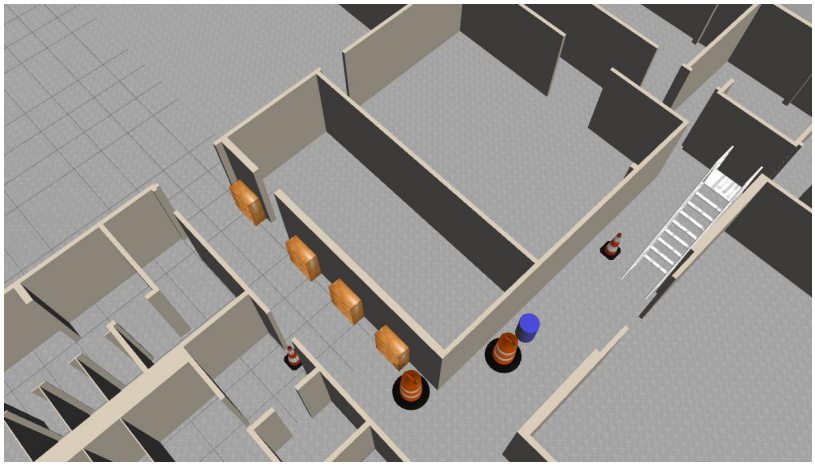


Figure 4.7: This picture depicts the environment used for the $1st$ simulation. Note how there are less objects in the scene then when the $2nd$ and $3rd$ simulation were run. This is because in the $1st$ simulation, the SAD and the $BWSAD$ methods both performed near perfectly.

Table 4.3: Results of the 3 simulations

	Percentage of successful runs out of 1000	
	Baseline	Learning Method
$1st$ Simulation	99.4%	100%
$2nd$ Simulation	87.3%	100%
$3rd$ Simulation	70.1%	79.5%

The results above show that for all three simulations, the learning method was able to achieve a greater percentage of success. The first simulation had results that were similar, with percentages of 99.4% and 100% for the baseline and learning method respectively. This is due to the high success rate of the baseline algorithm. To further examine the disparity between the two methods, more objects were added to the scene in order to produce a lower accuracy for the baseline as to allow the learning method more room to show further improvement in robustness. This is apparent in the results obtained from the $2nd$ simulation. Furthermore, the $3rd$ simulation is a result of a change in the starting location of the robot, such that traversing underneath the two tables was made more difficult. The increase in performance by the learning method shows that $BWSAD$ is more robust, than SAD, when encountering obstacles that obstruct the original path.

4.5 Understanding the increase in robustness

As a consequence of the previous results, we now know that the $BWSAD$ method has a higher success rate as compared to the baseline SAD method. As such, this section will focus on understanding why this is.

4.5.1 Why SAD is failing?

A failure analysis of the SAD method is conducted to determine what caused the failure. To do this we take a closer look at one of the scenarios in which the baseline SAD algorithm failed. Specifically we look within the $2nd$ simulation since this simulation shows the greatest disparity between the two methods, and thus has the most to offer in terms of understanding. The points of failure for the $2nd$ simulation are shown in Figure 4.8, where the red dots represent the end location of all the 127 failed runs of the SAD method.

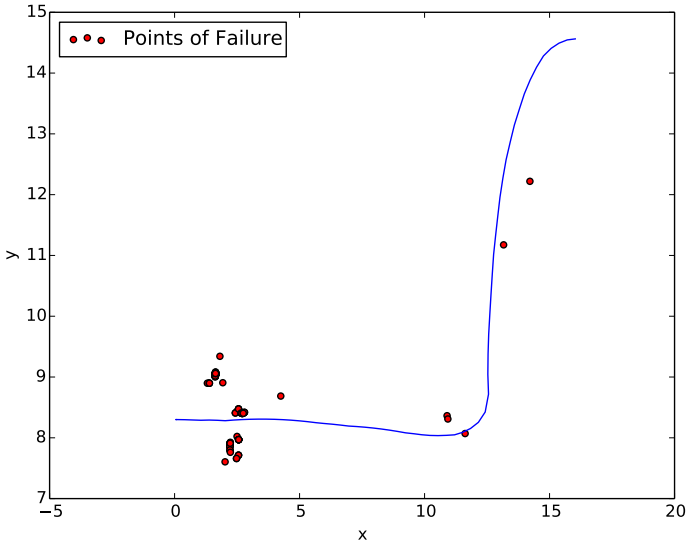
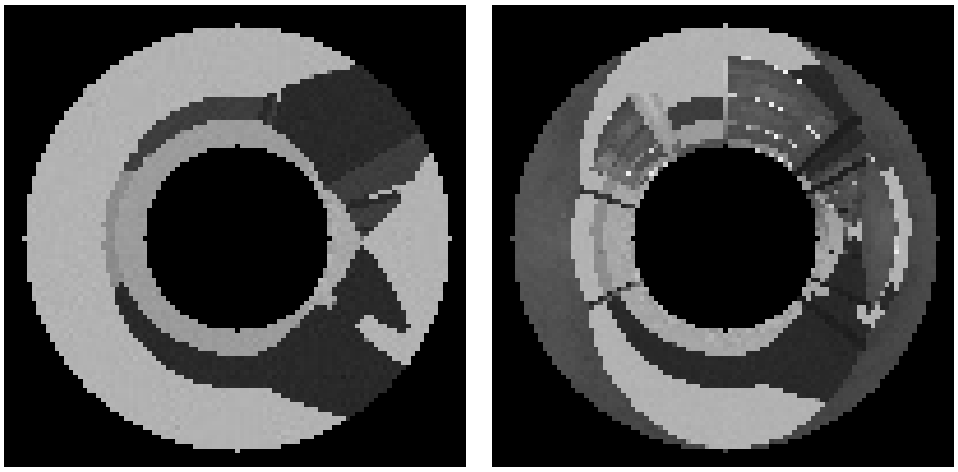


Figure 4.8: This Figure shows the training path taken by the Turtlebot, represented by the blue line, and the points at which the Turtlebot failed, which are represented as red dots. The Turtlebot started at the bottom left of the path and ended at the top right.

The greatest points of failure occurred close to the beginning of the path around the $10th$ training image. As such, Figure 4.9 shows images of the $10th$ training image, along with the view from the robot as it autonomously traverses the path close to the $10th$ training image. Notice how, with the exception of a few regions, the images look significantly different. This does not bode well for the SAD method, as it relies on the ability to compare these two images as similar. To determine how much of a problem this is, a general outline of how the robot navigates is provided.



(a) Original (b) Current Image

Figure 4.9: This figure shows the $10th$ training image (a) and current image (b) that is closest to the location at which the $10th$ training image was taken. The difference the new objects make is apparent.

To navigate the robot must determine the desired heading, θ, relative to its current body frame along with the index of the training image that it is closest to, denoted as k. The index that the robot believes it is at, is denoted as $\hat{k}$, also known as the perceived location. To determine the desired heading, the robot takes the current image, and rotates it 1°, 360 times for a total of 360°. Then all 360 of the rotated images are compared to the training image, $\mathbf{I}_0[\hat{k}]$, which is the training image believed to be closest to the current location. Figure 4.10 shows this heading comparison for the images in Figure 4.9(a) and 4.9(b). The lowest difference metric determines the desired heading. It is expected that the lowest difference should occur at a heading of 0 or very close, since the images were taken at roughly the same location but not exactly. As expected Figure 4.10 shows that the images are most similar when it is only slightly rotated. This shows evidence that SAD is robust to a large amount of variation in the environment.

Next, consider one of the autonomous runs taken during the $2nd$ simulation that failed close to the $10th$ training image. The heading comparison graphs for 4 locations close to the $10th$ training image are shown in Figure 4.11. Notice in Figure 4.11(b) that the heading values are all over the

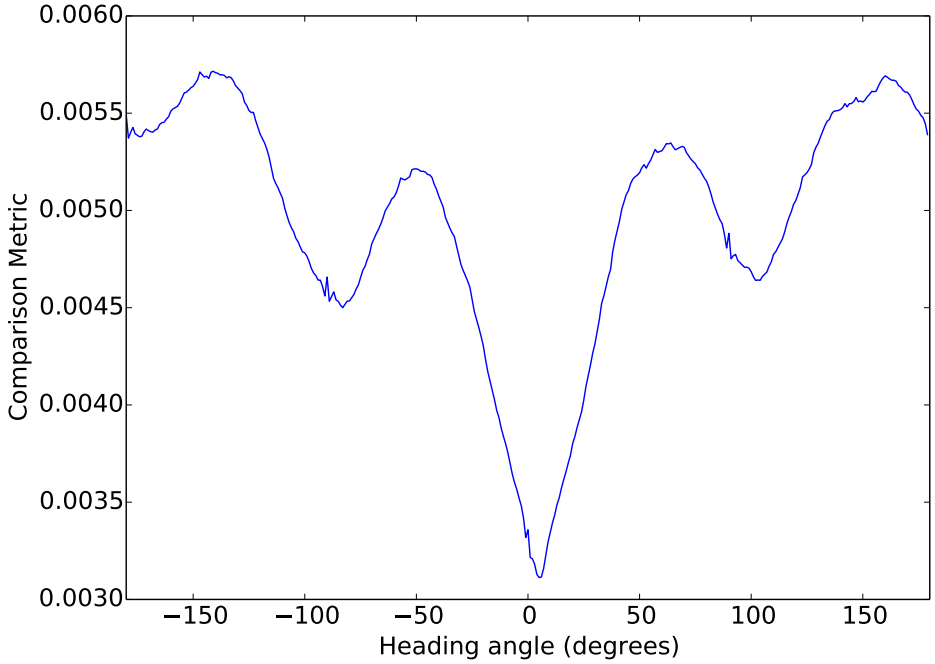
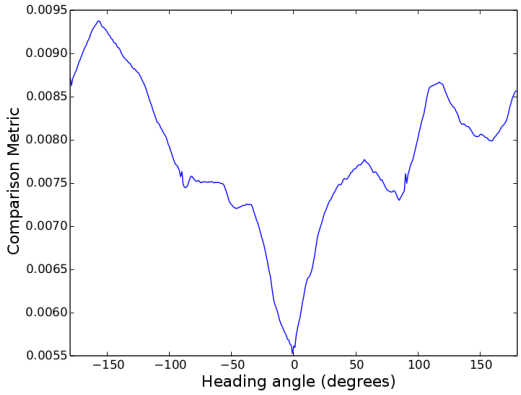


Figure 4.10: This figure shows the heading comparison of the images shown in Figure 4.9 where the x-axis is the amount that image 4.9(b) was rotated, and the y-axis shows the SAD comparison metric. A lower comparison means that the images are more similar.

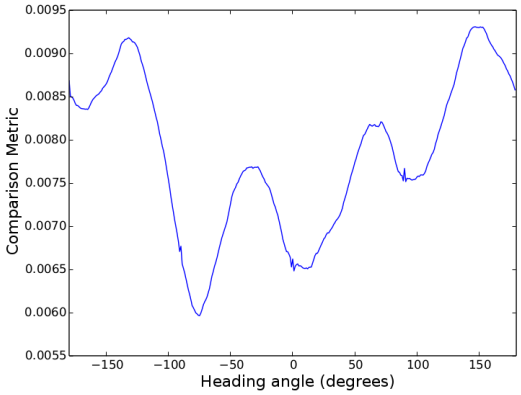
place, and the robot is lost. This is because the most similar match occurred at a negative heading, when the most similar match should have been at a heading of 0. This means the robot will turn left, however at this location the robot should be going straight. Moreover, this error never gets corrected, as by the next 2 comparisons 4.11(c) and 4.11(d) the robot still wants to turn left even though this is not correct. Even more concerning is that the previous Figure 4.10 has provided evidence that the added clutter alone is unlikely to be responsible for this discrepancy in the heading comparison.

Since the results of the heading comparison are inconclusive, the location comparison is looked into to see if there are any issues that are causing the error. When the robot compares the heading, it has a belief of what training image it is closest to, denoted by $\mathbf{I}_0[\hat{k}]$, and this training image is used to determine the desired heading. In order to determine $\hat{k}$, the robot first rotates the current image 12°, 30 times for a total of 360°. Note this rotation is different then the rotation 1° rotations used to determine the heading. Let the 30 rotated images be denoted by the set $\mathbf{R}$. Then the robot compares all the rotated image with the 4 training images $\{\mathbf{I}_0[\hat{k}], \mathbf{I}_0[\hat{k}+1], \mathbf{I}_0[\hat{k}+2], \mathbf{I}_0[\hat{k}+3]\}$ shown in equation (4.8).

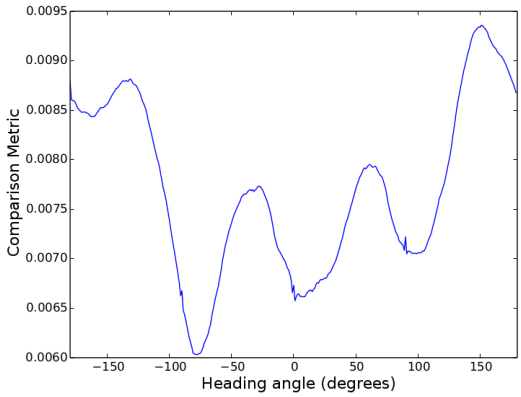
$$\mathbf{H}[i,j] = SAD(\mathbf{R}[\mathbf{i}], \mathbf{I}_0[j]) \tag{4.8}$$



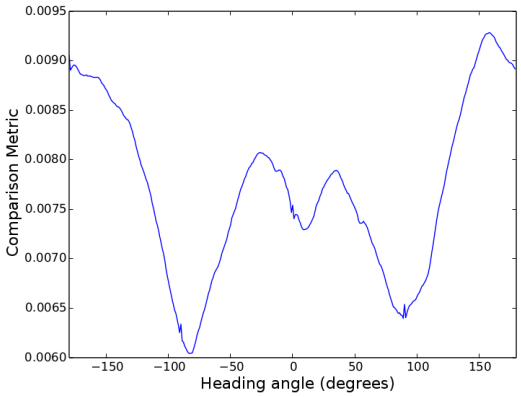
(a) Heading Comparison at the $20th$ time step



(b) Heading Comparison at the $21st$ time step



(c) Heading Comparison at the $22nd$ time step



(d) Heading Comparison at the $23rd$ time step

Figure 4.11: These figures show the heading comparison graphs for the autonomous run of the SAD method performed during the $2nd$ simulation. A lower comparison is favorable, and the lowest comparison determines the desired heading. Here the desired heading should be straight for all the graphs, however the $21st - 23rd$ time step graphs are setting the desired heading away from $0°$, which is not desired.

where $i \in \mathbf{R}$ and $j \in \{\hat{k}, \hat{k}+1, \hat{k}+2, \hat{k}+3\}$.

Next the minimum of $\mathbf{H}$ is calculated as in equation 4.9.

$$\mathbf{D}_j = \min_{i \in \mathbf{R}} \mathbf{H}_{i,j}, \tag{4.9}$$

In summary, the SAD method fails because of perceptual aliasing, which causes the robot to perceive its current location incorrectly. Since, the algorithm is not equipped to look backwards, these errors can compound upon each other and when the discrepancy between the perceived location and the actual location gets to be too large, the algorithm sends an incorrect turn signal. This leads to the robot getting lost and failing the run.

It is important to note that when the robot could also look backwards, the average performance

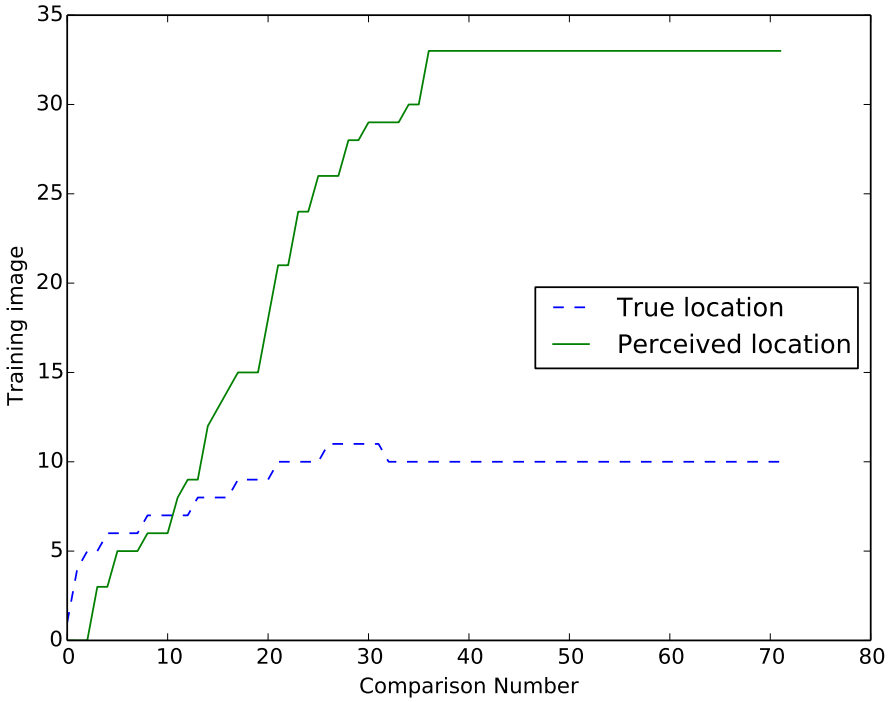
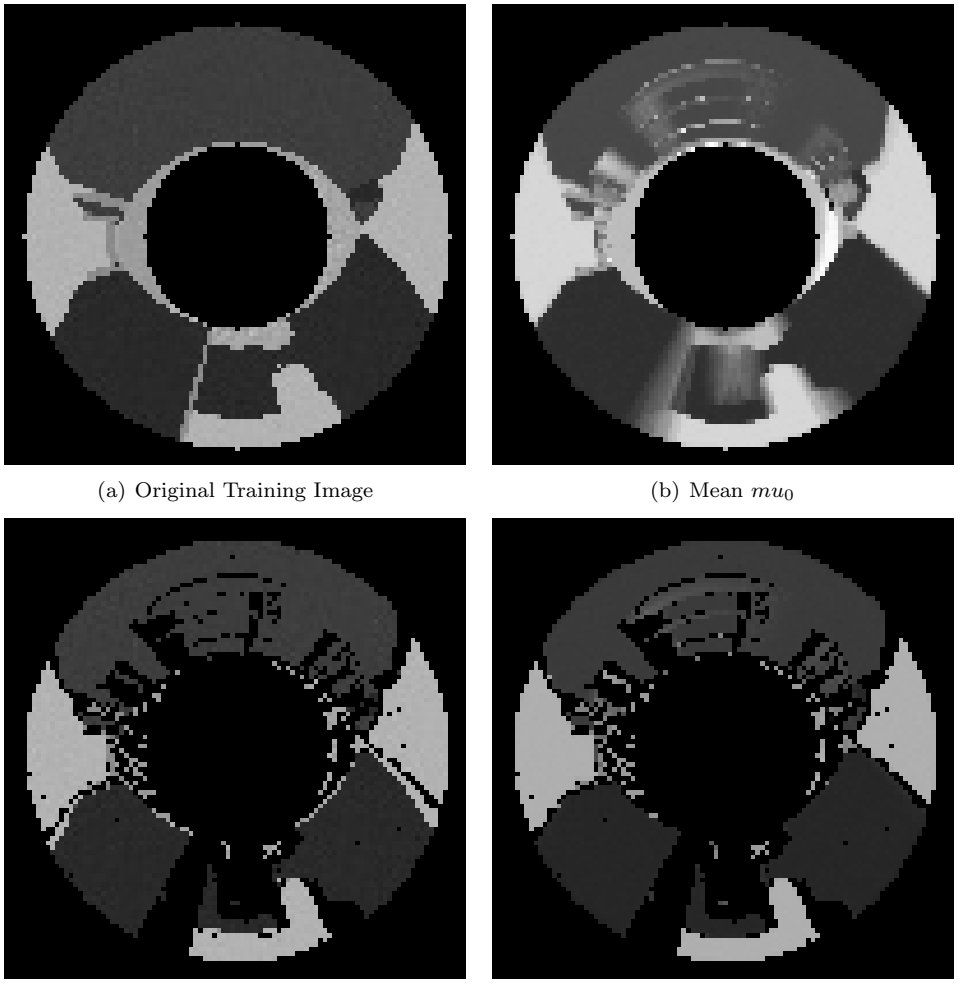


Figure 4.12: This figure shows the perceived location and the true location of the robot, relative to the training image locations. The failure of the SAD method can be seen at the $11th$ discrete time step. These lines should follow each other, however they diverge, showing that the robot thinks it is where it is not.

seen was a success rate at 40%. Forcing the robot to only look forward, caused an immediate increase into the 90% success rate area.

4.5.2 How BWSAD Adds Robustness

Now that it is understood how the SAD method is failing, we can look at how $BWSAD$ corrects this. The difference between SAD and $BWSAD$ is the weighting term that is applied to the standard SAD. To show how the weighting aspect of $BWSAD$ can improve performance, a sample of what the robot sees is shown in Figure 4.13. Notice how the original training image and the mean look quite different, however when the weighting is applied the blurred regions are covered, meaning that they are marked as unreliable and are thus not utilized for comparison.



(a) Original Training Image

(b) Mean mu_0

(c) Original Training Image plus Weighting

(d) Mean plus Weighting

Figure 4.13: This figure shows 4 different variations of the $30th$ training image. These variations are the original training image (a), the learned means (b), the original training image with the weighting term (c), and the learned mean with the weighting term (d). Notice how the top two images look quite different, and the bottom two images look similar. The pixels that are blacked out, are not used for comparison when using the $BWSAD$ comparison metric.

Since the point of failure for SAD is perceptual aliasing in the location determination; an analysis of this is a good place to go from here. For this, Figure 4.14 was created, which shows the perceived location and the true location, similar to Figure 4.12, except with $BWSAD$ instead of SAD. This graph shows that $BWSAD$ can track the location without getting lost unlike the SAD method.

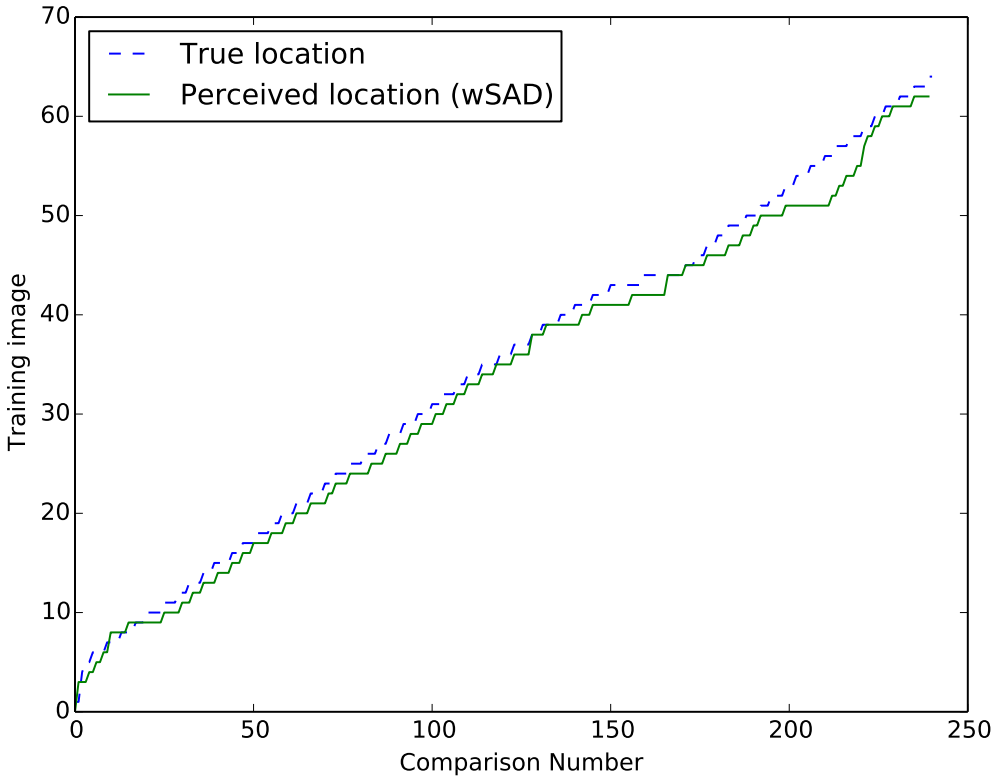
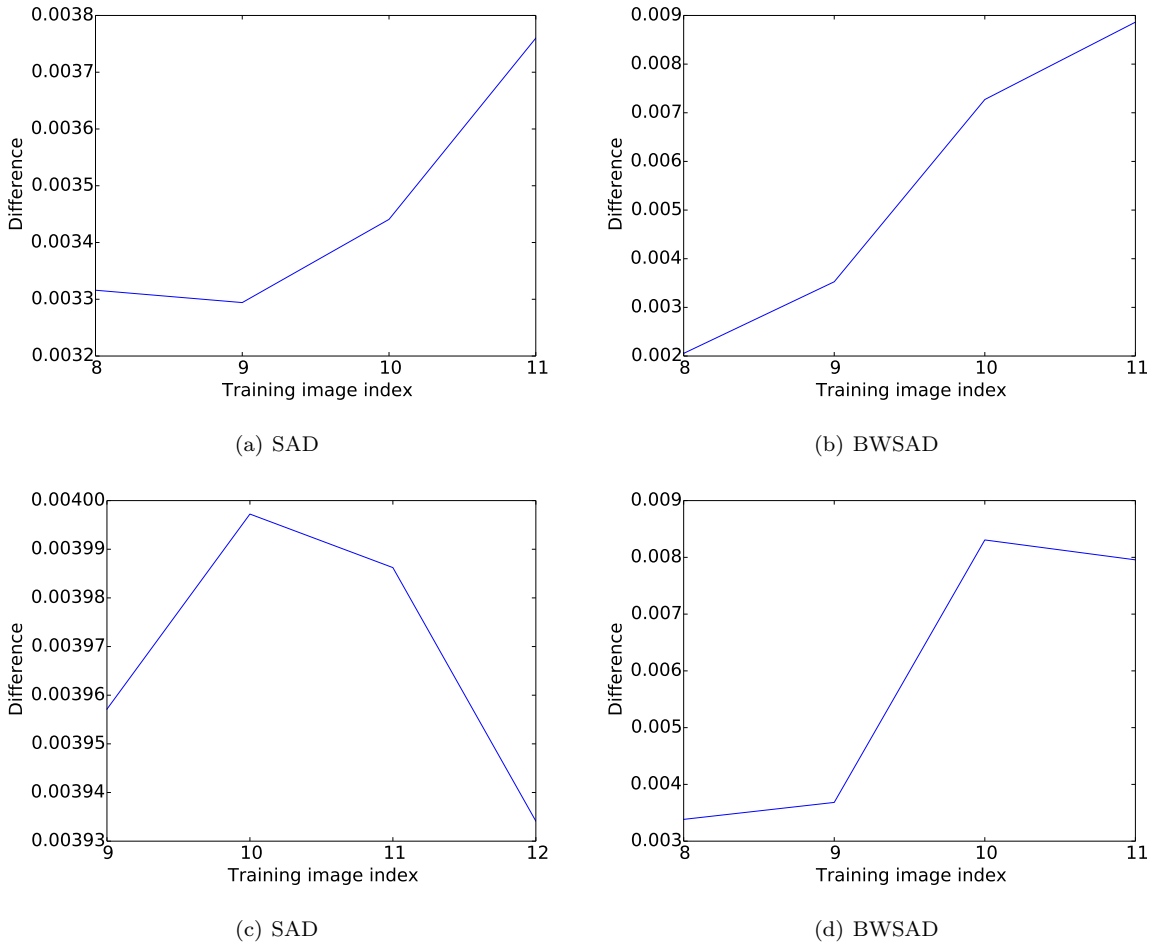


Figure 4.14: This figure shows the perceived location and the true location of the robot, relative to the training image locations. The data was obtained during an autonomous run of the $BWSAD$ method for the $2nd$ simulation.

From Figure 4.12 it was learned that the SAD method failed at the $11th$ instance of the heading update. Delving into this further, Figure 4.15 shows the location comparisons made by the robot for both SAD and $BWSAD$ and the $12th$ and $14th$ instances of the heading update. Both SAD and $BWSAD$ were compared as close together as possible; to be specific the top two graphs for SAD and $BWSAD$ were taken at the (x, y) location of $(.950, 8.294)$ and $(.930, 8.295)$ respectively and the bottom two were taken at $(1.112, 8.291)$ and $(1.128, 8.291)$ respectively. The true location of the top two graphs is the $7th$ training image, and the true location of the bottom graphs is the $8th$ training image. The results from Figures 4.15(a) and 4.15(c) show that SAD compares both locations incorrectly and Figures 4.15(b) and 4.15(d) show that $BWSAD$ compares one location as

ahead by one and the other it finds correctly.



(a) SAD

(b) BWSAD

(c) SAD

(d) BWSAD

Figure 4.15: This figure shows the values of the location determination from Algorithm 6 at 2 consecutive locations. In these graphs, a lower difference is considered to be better, and the lowest difference is the best location.

In summary, $BWSAD$ is able to achieve more robust navigation performance then SAD, because $BWSAD$ learns to focus only on regions within the training images that are less stochastic. This allows $BWSAD$ to overcome perceptual aliasing even when the original scenery has changed.

4.6 Real World Testing

Although simulations are great for obtaining Monte Carlo runs, they are often doomed to succeed since they tend to simplify the real world. This is especially true for vision based algorithms since 3D modeling has to be simplified to allow for real time rendering. Because of this, real world test have been performed to show how effective $BWSAD$ can be in a real world environment subject to stochastic changes. This section outlines and explains those experiments.

4.6.1 Robot and Software

The robot used for the real world experimentation was the Clearpath Robotics Turtlebot 2, with a Kobuki mobile base, Figure 4.16; the same robot that was used during the simulation tests. The Turtlebot comes standard with a Microsoft Kinect depth sensor, and a net book, which sports an Intel i3-4010U processor, and 4 Gbs of memory. A 360° catadioptric omni-directional camera was mounted to the top of the Turtlebot to allow for a full 360° camera view.

Figure 4.16: Image of the robot used for the real world experiments, the Turtlebot 2. The device on top is a catadioptric omni-directional camera.

The Turtlebot was programmed in Python with the help of ROS to interact with the Turtlebot. Given both the portable nature of ROS and the fact that the simulations were also done using ROS, the code used during the simulation was migrated to the laptop on the Turtlebot. Also, the computer vision library known as OpenCV was used to perform image processing based tasks, because OpenCV is well optimized and gives faster performance then relying on basic Python functions. To further optimize the algorithm, the images from the camera were down sampled to a resolution of 100×100. This was found to be the lowest resolution determined during the simulation testing, and thus was used for the real world testing.

4.6.2 Ground Truth

When running experiments indoors, the robot uses the adaptive Monte Carlo localization (AMCL) ROS package for the Turtlebot, to provide the ground truth location data. This localization method utilizes the on board Kinect sensor's depth readings along with odometry data combined in a particle filter to provide localization information.

4.6.3 Experimental Setting

The indoor experiments take place on the 2nd floor of the math and science building at Oklahoma State University. The room that the majority of the testing was performed in is known as the Robot Cognition Laboratory, Figure 4.17(a). The robot was also navigated through hallways, shown in Figure 4.17(b), to test the performance when given a plain environment in which to navigate.

(a) Robot Cognition Lab

(b) Corridor

Figure 4.17: Picture of the Robot Cognition Laboratory (a) and the hallway that provided the environment for the indoor testing (b).

Given the stochastic nature of robotic navigation, small changes in the starting location can have a large effect on how the robot navigates, often a few inches can result in success or failure. As such, to facilitate consistent initial conditions, a piece of tape was cut into an arrow shape and was placed on the ground. Subsequently, an arrow was put on the back of the Turtlebot, and when the robot is placed at the starting location for each experiment, both arrows were lined up to provide the desired consistency, Figure 4.18. This ensured that different starting positions played as small of a factor as possible in the results obtained.

Figure 4.18: Picture showing the starting condition of the robot. Note the how the arrow on the ground and Turtlebot are aligned to ensure consistent initial conditions

4.6.4 Corridor Testing

Training Run

For the first indoor test, the robot was tele-operated along the path, shown in Figure 4.19, using a wired Xbox 360 controller. Whilst the robot was tele-operated images were recorded from the omni-directional camera at a rate of 1 frame per second, illustrated by the dots in Figure 4.19. Due to the computationally intensive nature of AMCL the Turtlebot's position was only able to be updated every .5 meters, any smaller and the processor struggled to handle running AMCL and the navigation algorithms. The large concern of these experiments was whether or not the Turtlebot was able to follow the path and not how precisely the Turtlebot followed the path. So, the inaccuracy caused by the .5 meter update is of no concern.

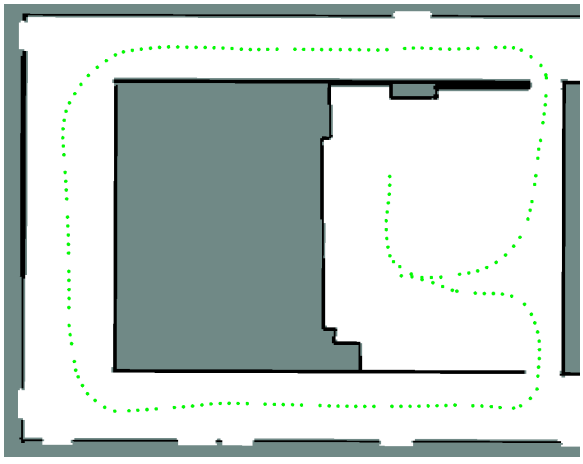


Figure 4.19: Picture of the training path taken through the lab, out to the hallway, and back into the lab. The dots represent the locations at which the training images were taken.

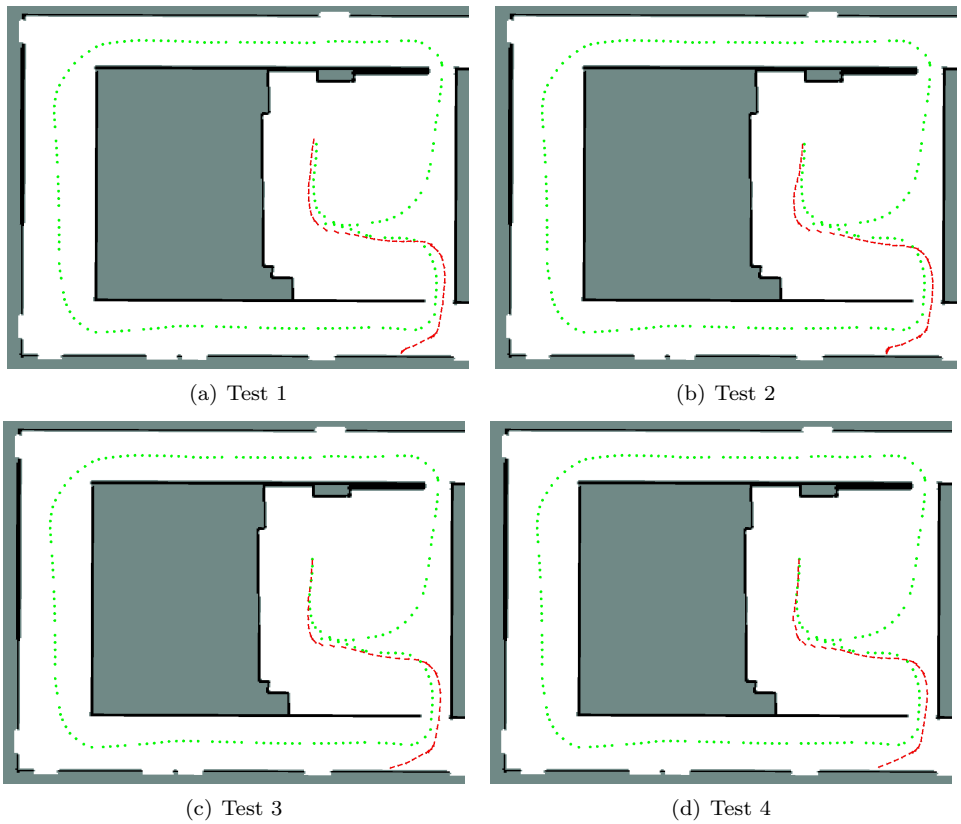
Autonomous Runs

For the first test, the robot compares the set of training images with the current image to determine the desired heading, by using the unlearned SAD comparison method. This navigation methodology is outlined during the simulations results and in Algorithm 1. The parameters that were used for this test are shown in Table 4.4. During the simulation experiments $\Delta\theta$ was set to 12, however it was increased to 30 for these tests.

Table 4.4: Parameters used for indoor testing

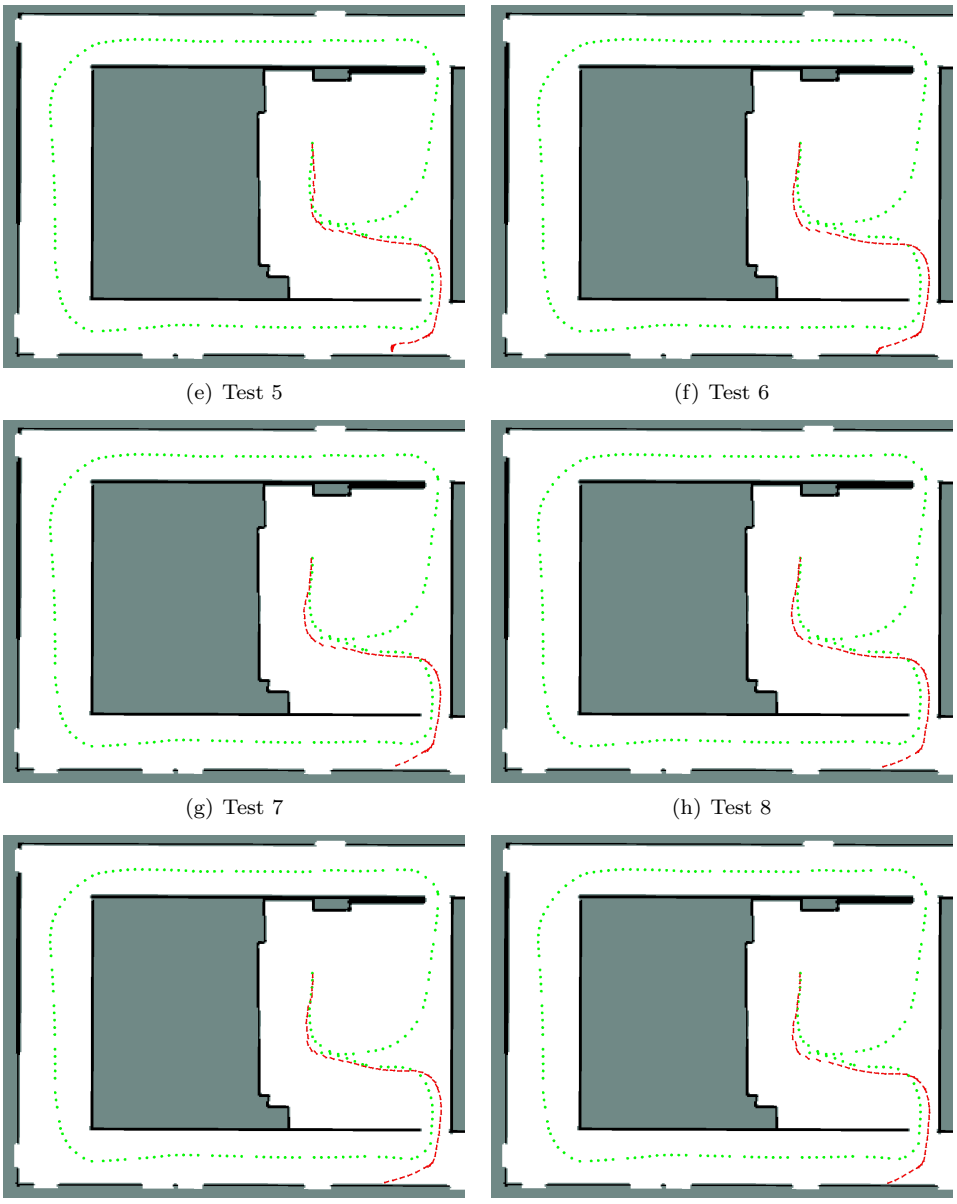
	$\Delta\theta$	Δk	p
Values	30	4	8

The robot retraced the original training path, Figure 4.19, a total of 10 times, and the results are shown in Figures 4.20. In these figures, the dots represent the location of each training image, of which there are 190 in total, and the arrows represent a vector defining the robots location and orientation at standard intervals of .5 meters.



(a) Test 1 (b) Test 2

(c) Test 3 (d) Test 4



(e) Test 5 (f) Test 6

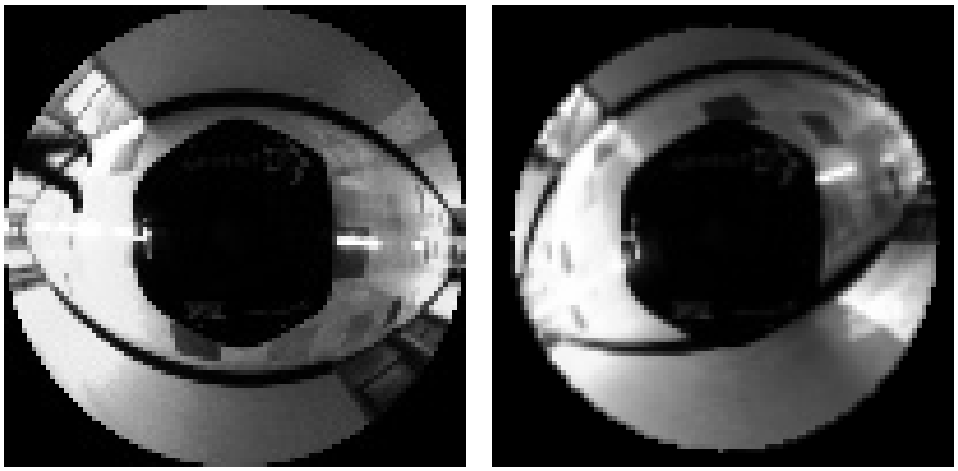
(g) Test 7 (h) Test 8

(i) Test 9 (j) Test 10

Figure 4.20: This figure depicts the 10 autonomous test runs of the unlearned SAD method. The dots represent the locations of the original training images, and the arrows represent the location of the robot while autonomously navigation. Notice how at the bottom, just after the robot leaves the door, the robot drifts to the left and hits the wall.

Note how the arrows deviated from the original path after the robot entered the corridor until the robot hit the wall. This indicates that the robot was unable to travel along the original training path correctly; as a matter of fact, the robot failed all 10 runs. When looking at where the robot thought it was, it appears that the robot was able to localize correctly for the majority of the path. Since, from Figure 4.20, it can be seen that the robot did not have a problem with localizing; it can be surmised that the problem lies within the heading calculations.

To analyze the heading calculations we look at two images. These images are Figure 4.21(a), the $44th$ training image, and Figure 4.21(b), an image taken during the $10th$ autonomous run when the robot was close to the $44th$ training image. The $44th$ image is used for this analysis because this image is where the robot thought it was right before hitting the wall. The robot determines its desired heading by first rotating the current image 1° 360 times. Next, it uses the SAD metric to compare all 360 rotated images to the training image that best represents the current location of the robot as determined by Algorithm 6. The rotation amount that best corresponds to the lowest comparison indicates what the desired heading is.



(a) 44th Training Image (b) Current Image

Figure 4.21: Images of the $44th$ training image, (a), and the current image, (b), that compared the best during the unlearned test.

As a human, it is easy to see that the image taken during the autonomous retracing, Figure 4.21(b), should be rotated clockwise $15 - 20^\circ$ to best match the orientation of the training image, Figure 4.21(a). However looking at Figure 4.22, it can be seen that the SAD comparison metric sees the images as the most similar with very little rotation. This led the robot to set the desired heading to -1 and continued to go straight. Shortly thereafter the robot hit the wall and failed the navigation, because it was never commanded to turned away from the wall.

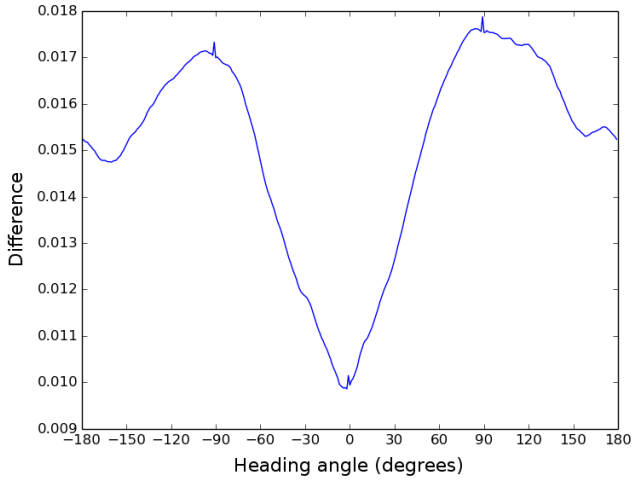


Figure 4.22: This graph depicts all 360 comparison values, using the unlearned SAD comparison metric, between the $44th$ training image, and all 360° rotations of the image shown in Figure 4.21(b). This graph is how the navigation scheme in this thesis determines the desired heading, by turning until the global minimum is at 0.

Next, the learned $BWSAD$ method was tested to see if it could retrace the path correctly, where the unlearned version could not. However, since the unlearned test failed, the training set cannot be updated using this data because learning off of failed data would be undesirable. Instead a manual retracing of the path was performed to update the posterior distributions, which in turn defines the means, μ, and the weights, $\mathbf{W}$, of each training image used for the $BWSAD$ comparison metric. This was done by tele-operating the robot along the path and updating the means and the weights in the same manner as is described in Algorithm 4. This manual retracing is shown in Figure 4.23 below.

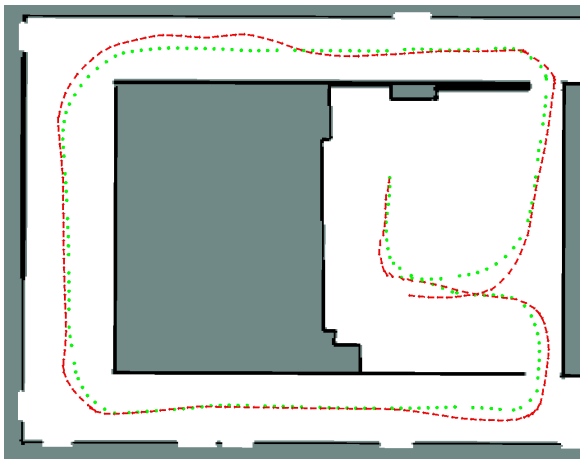
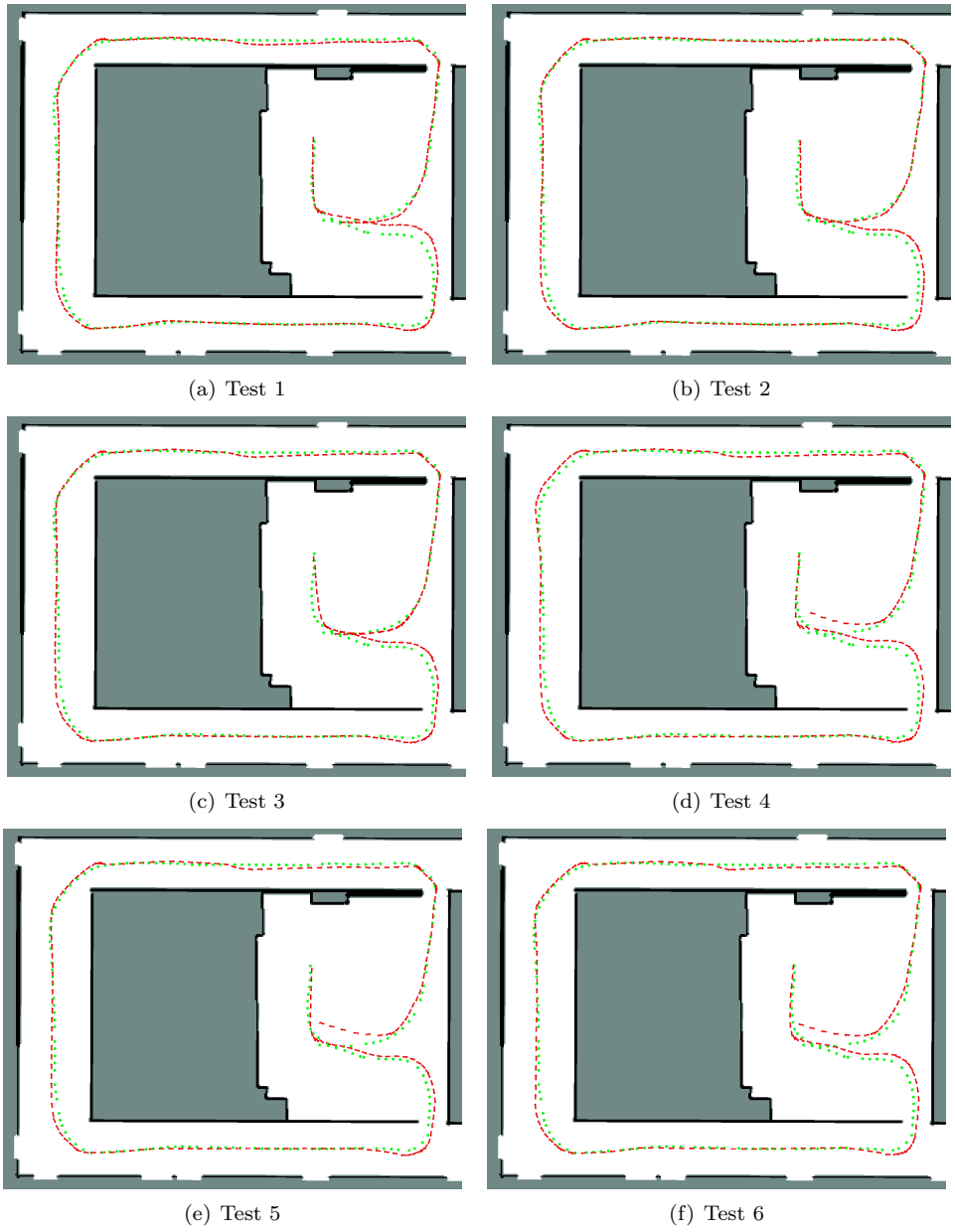


Figure 4.23: This figure depicts the manual retracing of the path done to update the posteriors obtained from the original training run.

With the posterior training set updated across the entire path, the next set of runs could be performed. These runs consisted of 10 autonomous retracings of the original training path, Figure 4.19, using the newly updated posterior distributions from the manual retracing, Figure 4.23. Figure 4.24, shows these 10 autonomous runs.



(a) Test 1

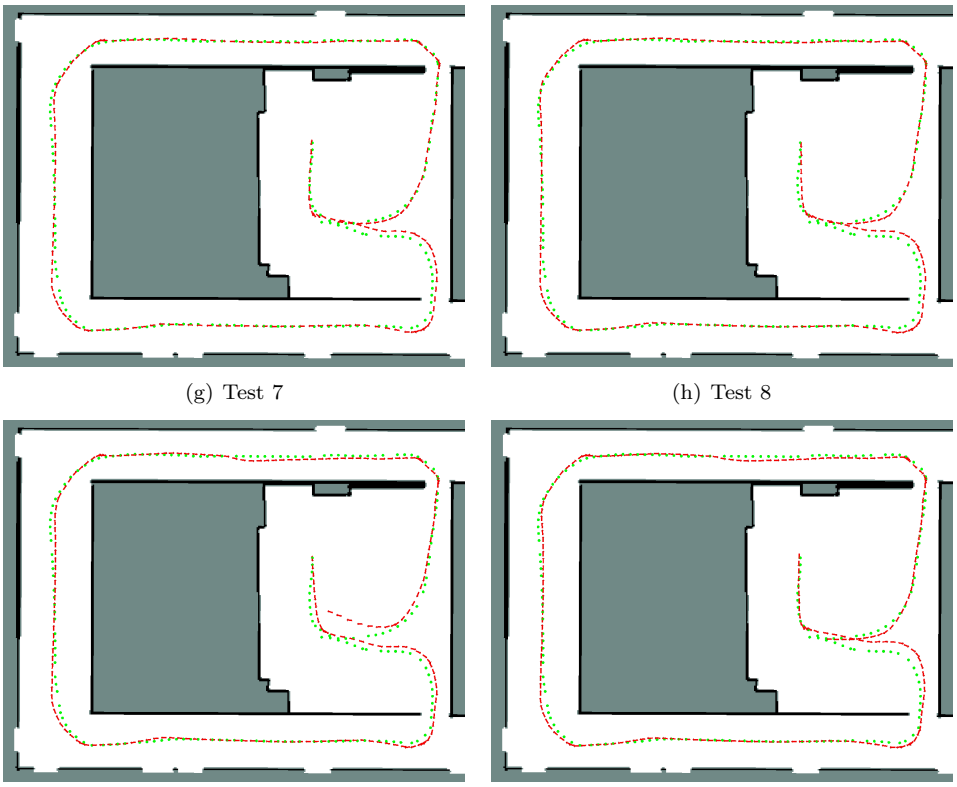
(b) Test 2

(c) Test 3

(d) Test 4

(e) Test 5

(f) Test 6



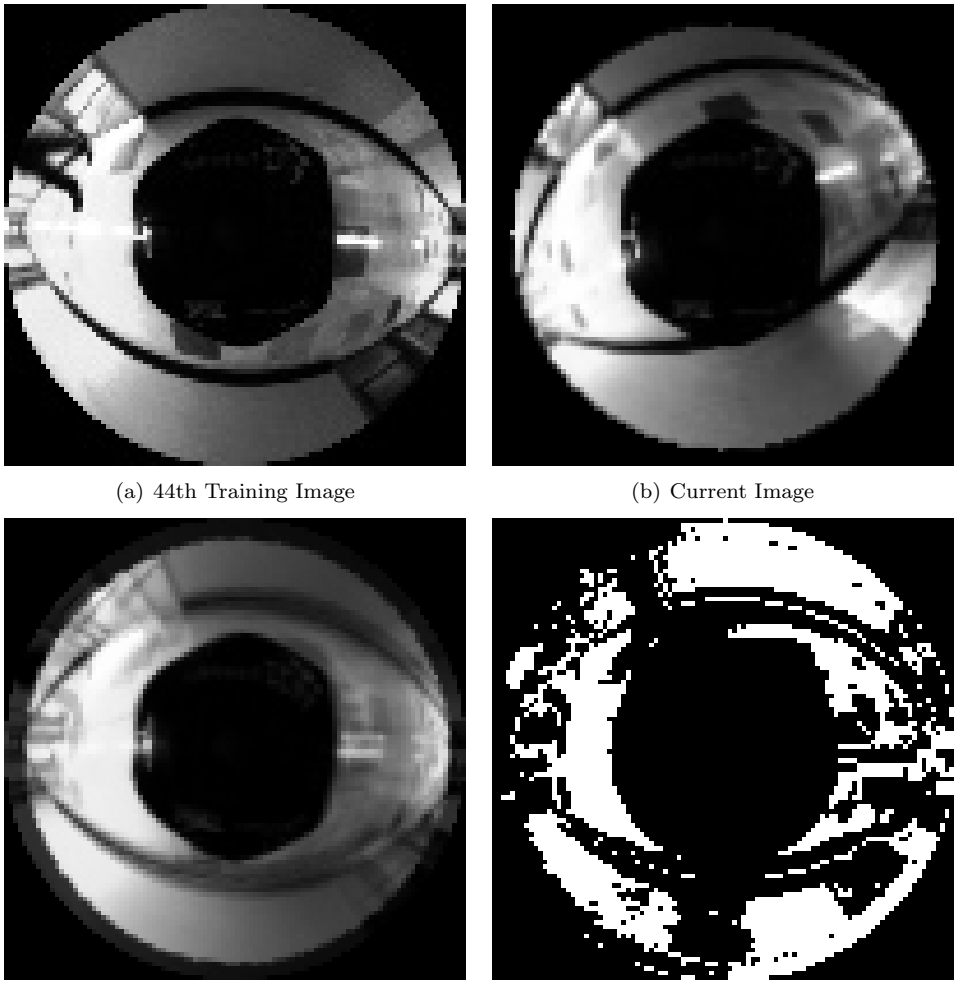
(g) Test 7 (h) Test 8

(i) Test 9 (j) Test 10

Figure 4.24: This figure depicts the 10 autonomous test runs of the learned $BWSAD$ method during the corridor testing. The dots represent the locations of the original training images, and the arrows represent the location of the robot while autonomously navigation.

From the previous figures, it is clear, when using the updated posteriors, the $BWSAD$ method is able to complete the path 10 times out of 10. This is a significant increase in the reliability of navigation when compared to the previous unlearned runs. To understand how $BWSAD$ achieved this increase in performance, Figure 4.25 shows the same 2 images analyzed in Figure 4.21 along with the corresponding means, μ, and weights, $\mathbf{W}$, learned during the manual retracing.

Most important of these 4 images, is the learned weighting images. Looking at the weighting it is clear that edges, such as the black outline on the walls right by the floor, are considered unreliable. This makes sense because small changes in the robots position can result in these edges not comparing correctly. This is most apparent in the updated mean image, where these edges look gray, because when the robot combined multiple images, sometimes these pixels were white and sometimes they were black. By averaging these pixels, the result was gray. Also, the weighting appears to be masking certain area, such as the doors, tiles on the ground, and a black square on the bottom of the image. This black square actually corresponds to a lighting issue, because the



(a) 44th Training Image

(b) Current Image

(c) 44th Updated Mean

(d) 44th Weights

Figure 4.25: This Figure shows 4 images associated with the $44th$ training image. These 4 images are the original training image 4.25(a), the current image best compared to the $44th$ training image 4.25(b), the updated mean after the manual updating 4.25(c), and the learned weights as a result of the manual updating 4.25(d).

lights above the robot are causing this area to be lit up brighter then before.

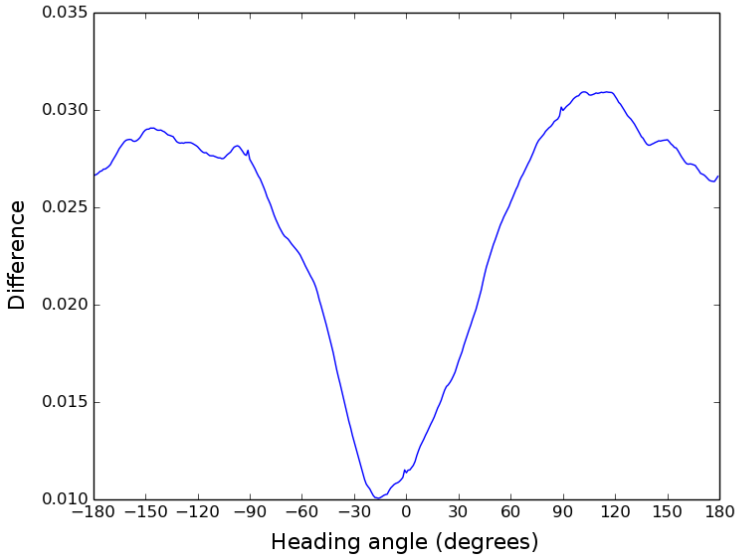
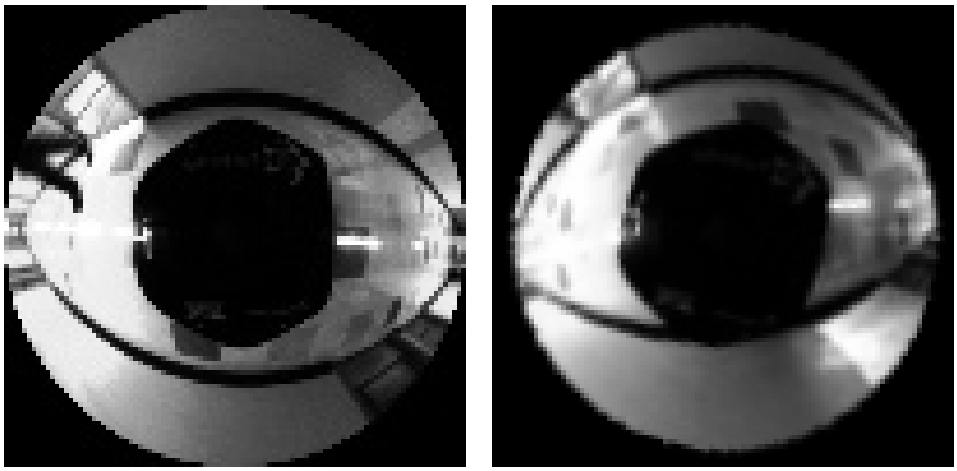


Figure 4.26: This graph depicts all 360 comparison values, using the unlearned SAD comparison metric, between the $44th$ training image, and all 360° rotations of the image shown in Figure 4.21(b). This graph is how the navigation scheme in this thesis determines the desired heading, by turning until the global minimum is at 0.



(a) Original Training Image (b) Current Image Corrected by BWSAD

Figure 4.27: These images show how by using the weights obtained from the manual retracing, $BWSAD$ is able to determine a suitable heading.

Figure 4.26 depicts the same graph as Figure 4.22. However, the comparison metric has been changed to the $BWSAD$ metric, which uses the means, Figure 4.25(c), and weights, Figure 4.25(d), learned from the manual retrace. From this graph, it can be seen that, by applying the learned weights from the manual retracing, the desired heading is set to 15°. This means that the robot will correctly navigate away from the wall as desired. To see the difference that the learned weights

make, the current image from Figure 4.26 was rotated by 15°, as was determined by the previous graph. The comparison of this rotated image and the original training image are shown in Figure 4.27. Now the image, Figure 4.27(b) looks as if it has been rotated to a similar orientation with that of the training image. This shows that, by using the learned weights, the robot is able to navigate correctly in a situation that the unlearned SAD comparison method was not.

4.6.5 Different Lighting Conditions

One of the most difficult problems for vision-based navigation is the stochasticity of illumination levels. Small lighting changes, that seem trivial to the human eye, can have a large effect on cameras. Illumination changes can cause an entire scene to look completely different, especially when the lighting changes are not uniform across the entire scene. Normally preprocessing technique, such as patch normalization, are used to provide illumination invariance. However by using $BWSAD$, the change in lighting can be learned. This allows $BWSAD$ to navigate the same path under different lighting conditions.

Training Run

The next experiment shows that the $BWSAD$ algorithm is capable of navigation in different lighting conditions, specifically, when half the lights are turned off. This testing took place in the Robot Cognition Lab and did not navigate through the corridor, because lighting conditions could not be controlled as precisely in the corridor as they could in the lab. First, a set of training images was obtained by tele-operating the robot with a wired Xbox 360 controller along the path shown in Figure 4.28. For this initial training run, all the lights in the lab were turned on.

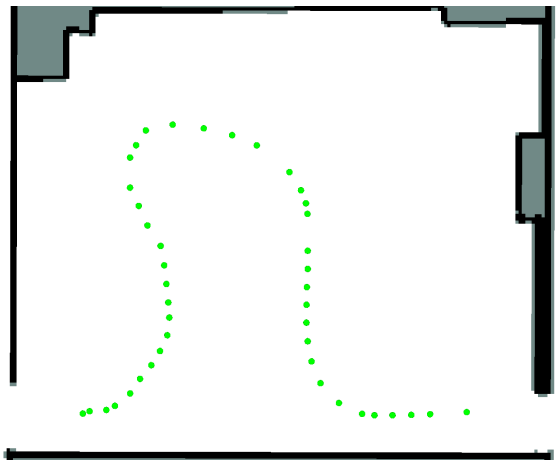


Figure 4.28: Picture of the training path, for the second indoor experiment, through the lab. The dots represent the locations of each training image.

Autonomous Runs

To accommodate a change in lighting, the back half of the lights in the room were turned off. This can be seen in Figure 4.29, where the dark shading on the top half represents the section of lights that were turned off. Also, Figure 4.30 shows two pictures of the lab with all the lights on, Figure 4.30(a), and the back lights off, Figure 4.30(b).

Figure 4.29: Map of the Robot Cognition Laboratory, where the dark top half corresponds to the half of the lights that were turned off.

(a) All lights on

(b) Back lights off

Figure 4.30: Pictures of the Robot Cognition Laboratory, in which the lighting was changed from all the lights on (a) to the back half of the lights off (b).

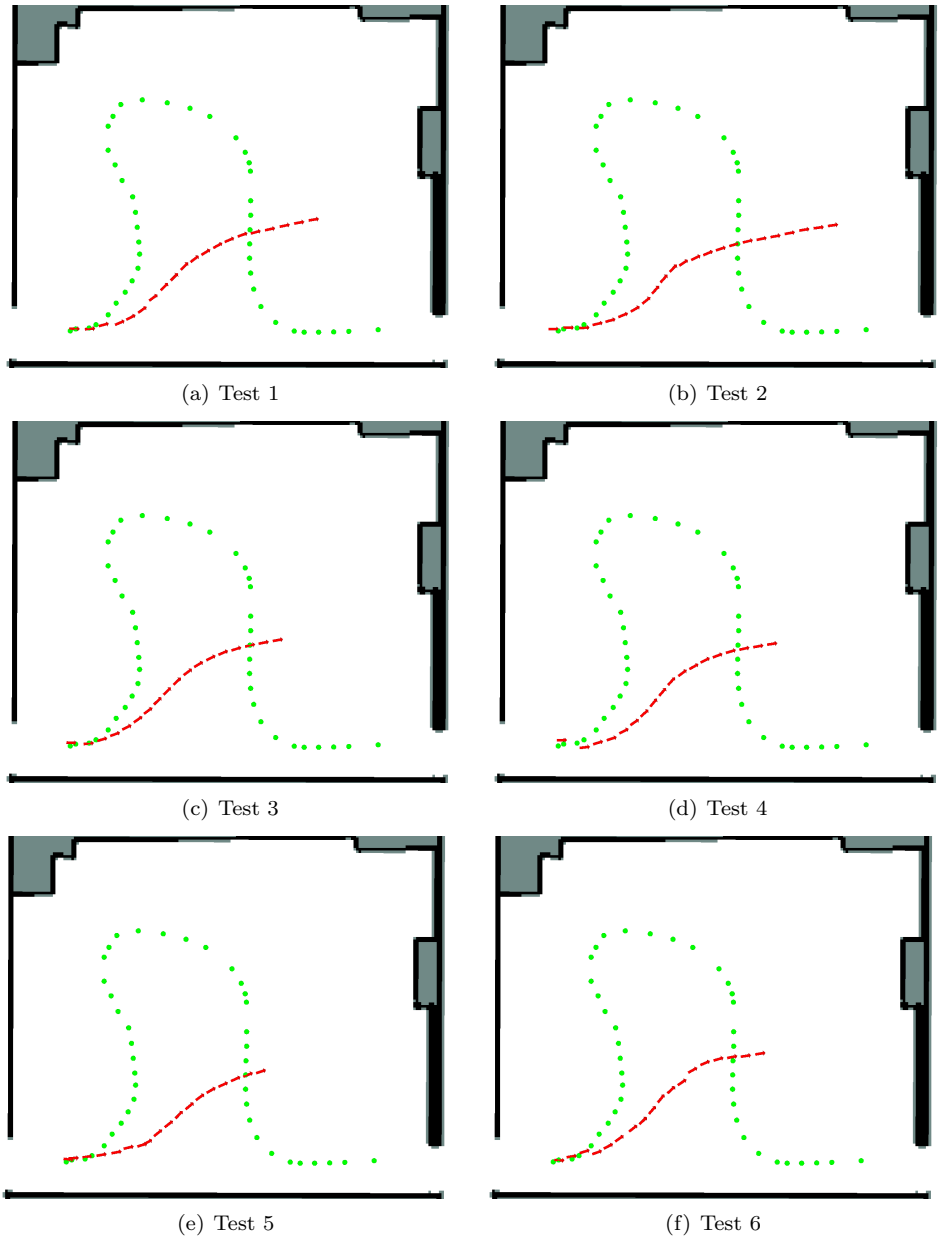
Table 4.5 shows the parameters, for Algorithm 1, that were used for this experiment.

Table 4.5: Parameters used for indoor testing

	$\Delta\theta$	Δk	p
Values	30	4	10

Note that the turn gain for the proportional controller, p, has been increase from 8 to 10, as compared with the previous indoor tests. This is because, the robot has to navigate a tighter path, so the turn gain was increased accordingly. Next, the robot used the unlearned SAD metric to navigate the original training path a total of 10 times. The results are shown in Figure 4.31. It is

important to note that the unlearned SAD metric is not expected to complete the path correctly because SAD does not have illumination invariance. Instead, these runs were done for comparison and analysis reasons.



(a) Test 1

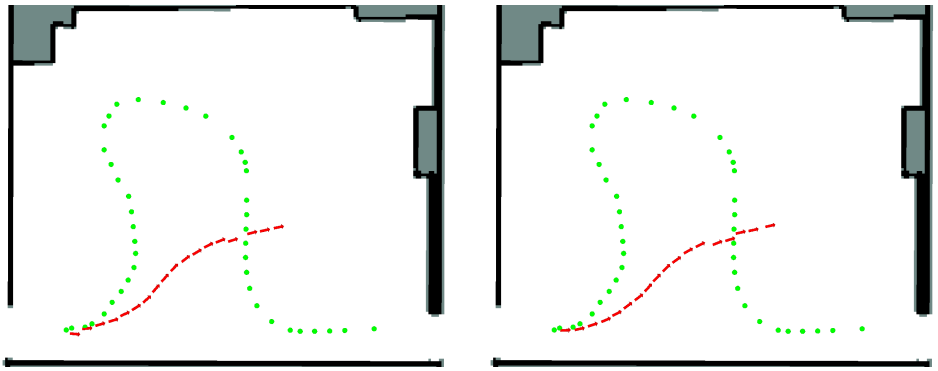
(b) Test 2

(c) Test 3

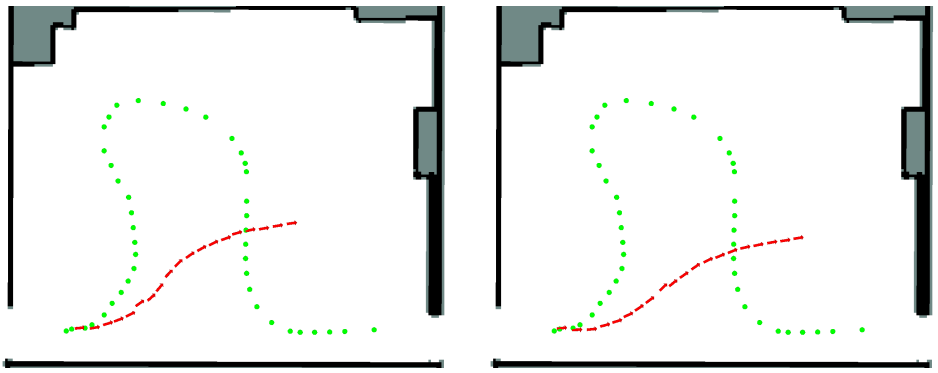
(d) Test 4

(e) Test 5

(f) Test 6



(g) Test 7 (h) Test 8



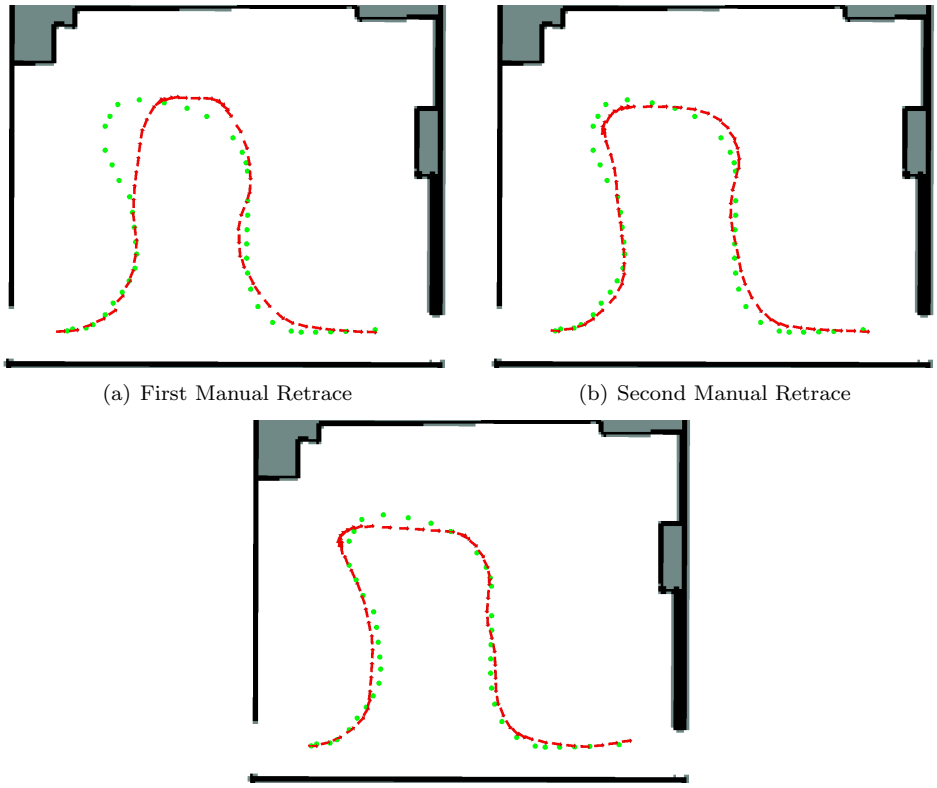
(i) Test 9 (j) Test 10

Figure 4.31: This figure depicts the 10 autonomous test runs of the unlearned SAD method when the back lights were turned off. The dots represent the locations of the original training images, and the arrows represent the location of the robot while autonomously navigation. Notice how the arrows trail off from the beginning and completely fail to follow the original path. The top half of the room is where the lighting was changed; thus showing that the unlearned method was unable to travel under different lighting conditions.

As can been seen from the previous runs, Figure 4.31, the robot could not remain on the correct path during the first turn and got stuck on a training image, the $10th$ training image to be exact. From these results, it appears that the robot had trouble navigating through the section of the room where the lights had been turned off. This is no surprise since the comparison metric, SAD, does not apply any normalization or illumination invariance techniques.

Next, it was desired to see if by learning the means, μ, and the weights, $\mathbf{W}$, the $BWSAD$ method could work in the different lighting condition. However, since the unlearned test failed, the training set could not be updated using this data. Instead a manual retracing of the path was required to update the posterior distributions. This was done by tele-operating the robot along the original path and updating the posteriors, in the same manner as was done in the previous corridor experiment. A total of 3 manual retraces were done, where the second retrace was performed with the lights on and the first and last were performed with the back lights off. This was done to prevent the posteriors

from being updated with the back lights off more then with all the lights on; thus, allowing the robot to travel with the back lights on or off. These retraces can be seen in Figure 4.32.



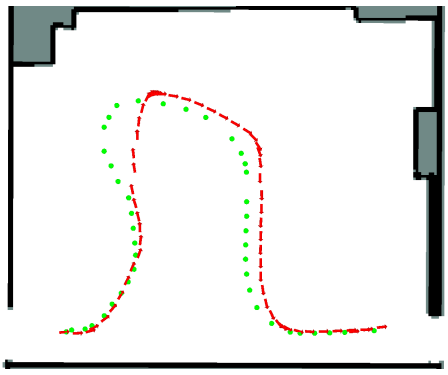
(a) First Manual Retrace

(b) Second Manual Retrace

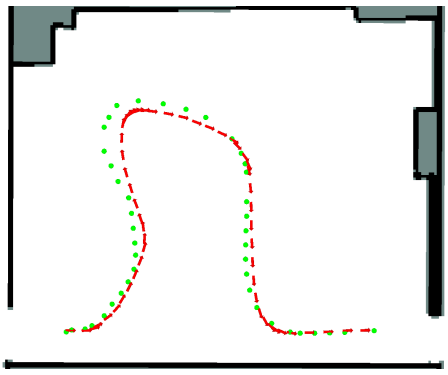
(c) Third Manual Retrace

Figure 4.32: This figure depicts the 3 manual retraces of the path that were performed to update the posteriors. The first and third, (a) and (c), were done with the back lights off, and the second, (b), was done with all the lights on.

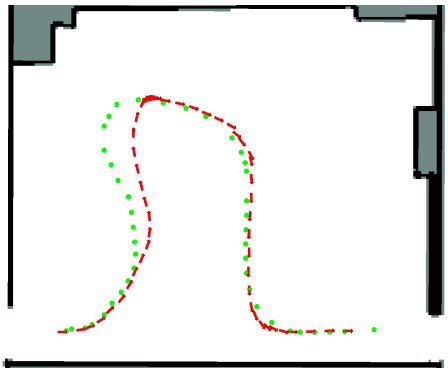
Next, the newly updated means and weights were used to navigate the robot over the original training path a total of 10 times. This was done while the back lights were turned off to provide the different in lighting. The results are shown in Figure 4.33.



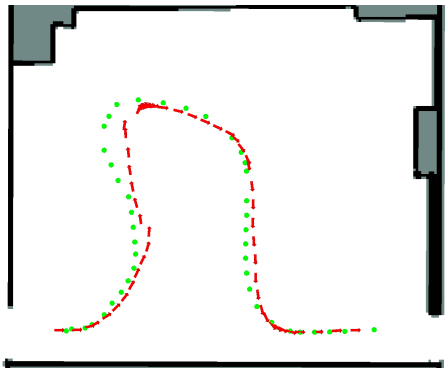
(a) Test 1



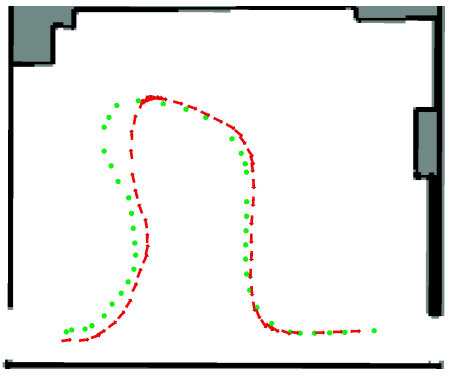
(b) Test 2



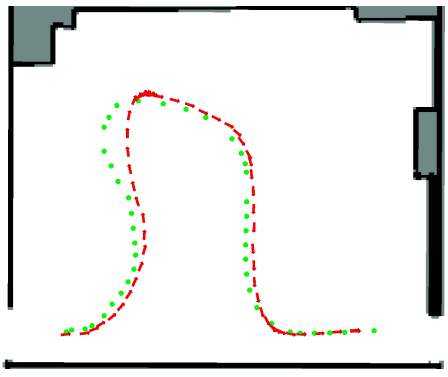
(c) Test 3



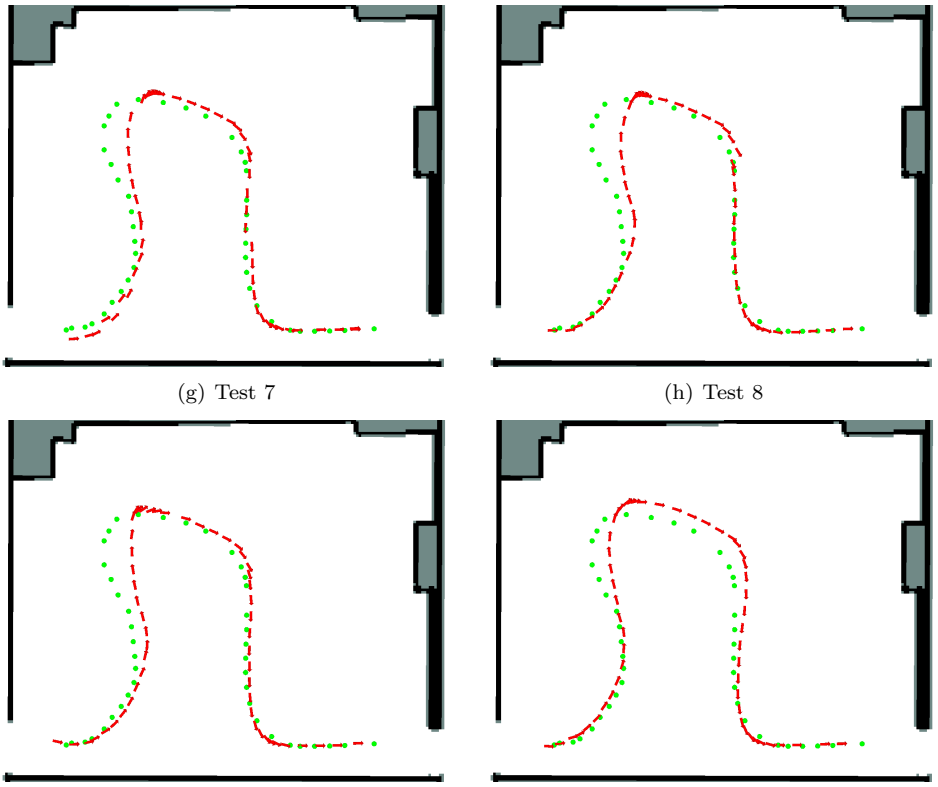
(d) Test 4



(e) Test 5



(f) Test 6



(g) Test 7 (h) Test 8

(i) Test 9 (j) Test 10

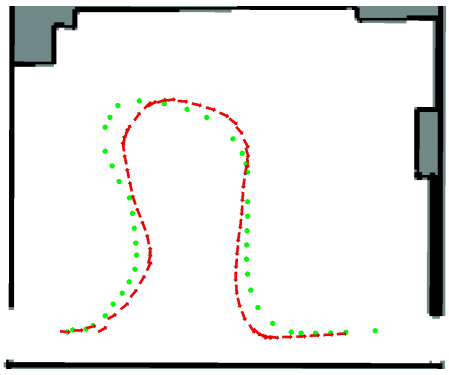
Figure 4.33: Results of the learned $BWSAD$ navigation method with the back lights turned off. The dots represent the locations of the original training images, and the arrows represent the location of the robot while autonomously navigating.

As can be seen from the previous runs, by using the learned means and weights the robot was able to navigate the new lighting conditions successfully all 10 times. Thus, even under different lighting conditions, the $BWSAD$ method was able to learn the new lighting conditions and navigate correctly. However, the purpose of these experiments was to show that $BWSAD$ could travel under multiple lighting conditions while using the same set of posteriors. It has only been shown that $BWSAD$ could travel under the new lighting conditions, but it is not known if $BWSAD$ can travel under the original lighting conditions after the posteriors were updated. Thus 10 more runs were performed with all the lights on, using the same updated posteriors from the previous runs. Figure 4.34 shows these runs.

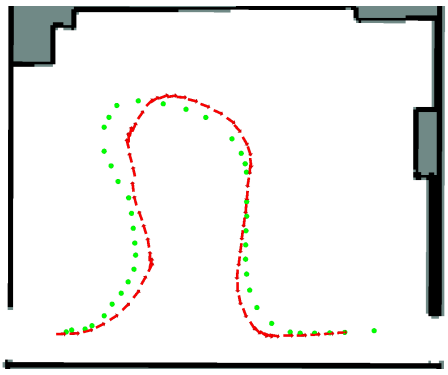
From the previous runs, it is clear that the updated $BWSAD$ metric was able to successfully navigate in the original lighting conditions 10 out of 10 times. This shows that even when the robot was trained with full light and then updated with half light, the robot was still able to successfully navigate under both conditions. To show how this works, Figure 4.35 shows the weights, $\mathbf{W}$, and

means, μ, for the $21st$ training image. Notice how in Figure 4.35(d) the top left of the image, which corresponds to the area with the lights on, is blacked out and deemed unreliable. At first this seems odd since this is the area which has the same lighting as the original training set. However, this is due to the aperture of the camera, which has opened up in the dark lighting, thus making the lit area brighter. This is most apparent in Figure 4.35(a), which depicts the $21st$ training image and Figure 4.35(b), which depicts an image saved from the $10th$ autonomous run that best compared with the $21st$ training image. Notice how the top left of the image is very bright, even though the light levels have not changed in that part of the room. Because the robot is in the dark half of the room, the camera has opened the aperture to let in more light, thus making the other half of the room, top left ot the image, brighter.

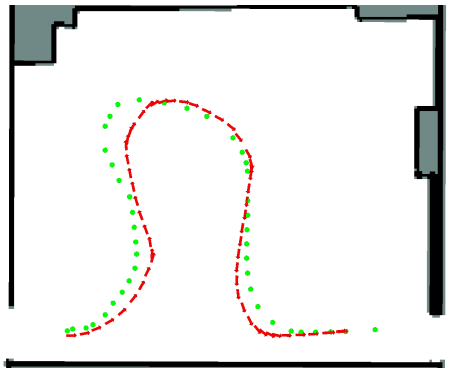
From this we can conclude that, by marking the segments of the scene that were the least effected by the change in lighting as reliable, $BWSAD$ is able to navigate the path correctly under both lighting conditions.



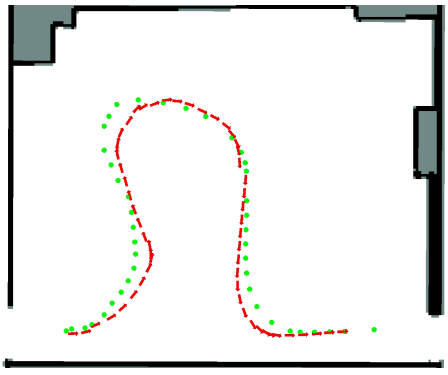
(a) Test1



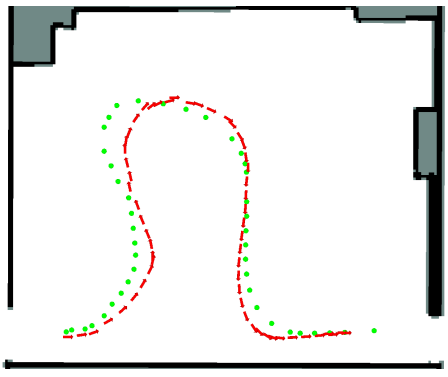
(b) Test2



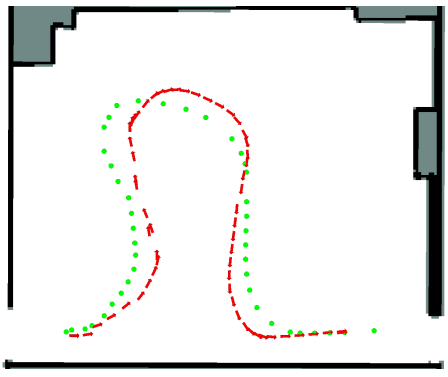
(c) Test3



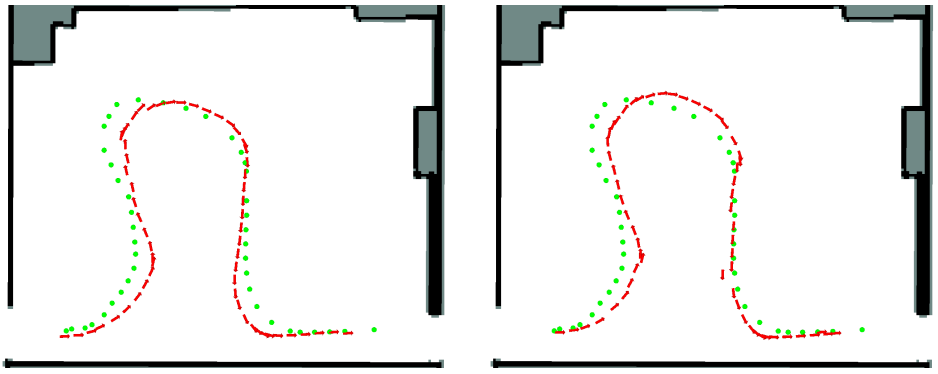
(d) Test4



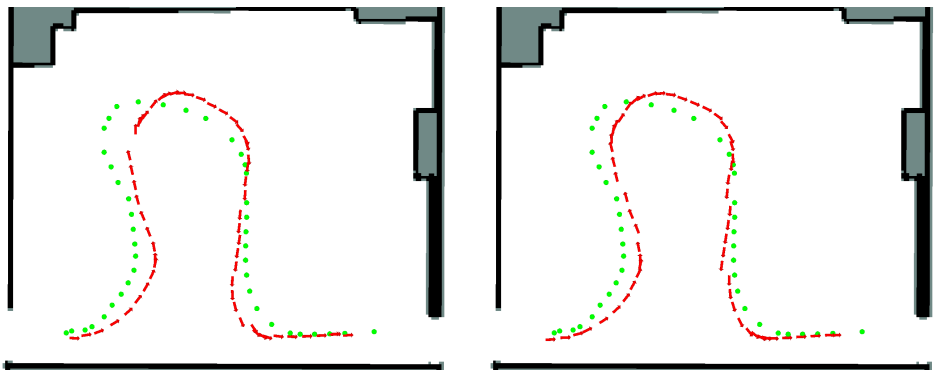
(e) Test5



(f) Test6

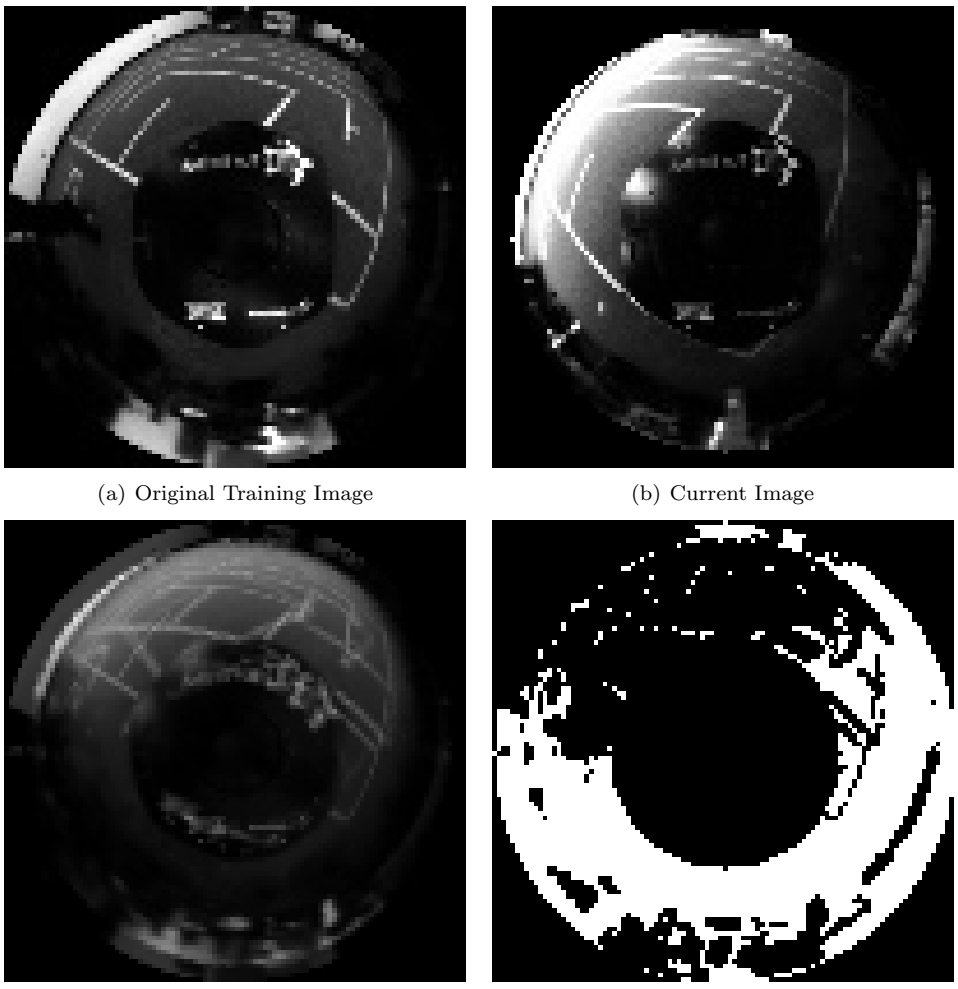


(g) Test7 (h) Test8



(i) Test9 (j) Test10

Figure 4.34: Results of the learned $BWSAD$ navigation method with all the lights turned on, after the Turtlebot learned how to travel with the back lights off. The dots represent the locations of the original training images, and the arrows represent the location of the robot while autonomously navigating.



(a) Original Training Image

(b) Current Image

(c) Mean

(d) Weighting

Figure 4.35: Images of the learned means and weights of the $21st$ training image, along with the original training image and the current image associated with the $21st$ training image. Notice what the lighting condition has changed between the original training image and the current image. The weights mark the location that corresponds to the lighting change as unreliable.

Chapter 5

Conclusion

5.1 Contributions and Observations

In this thesis, we have taken a simple visual memory navigation algorithm and have increased the performance under the presents of both scene changes and lighting changes. By applying Bayesian learning to the original set of training images, $BWSAD$ was able to learn regions within the scene that were not reliable for consistent visual navigation and only focus on those regions deemed reliable.

We have shown, through real world testing, our algorithm was in fact able to navigate two different lighting scenarios using one updated training set. The results further showed that this was accomplished through our algorithm's ability to learn what regions of the scene are similar across the different lighting scenarios. By focusing on these regions of similarity, our method was able to navigate in conditions that made navigation impossible without the learning aspect.

It has also been shown, through real world testing, that our algorithm was able to increase the robustness of a simple visual memory navigation method. This was accomplished through the Bayesian weights which were able to locate regions of low reliability through the variance of pixel values observed during different path iterations.

5.2 Future work

Although $BWSAD$ has shown good performance when navigating indoors, it has struggled to navigate in an outdoor environment properly during testing. To give $BWSAD$ the ability to navigate in outdoor environments, other template matching methods that provide a better form of illumination invariance, such as the normalized cross-correlation coefficient (NCC), could be used. However using such methods in tandem with the Bayesian weights described in this paper presents a host of possible issues, and is the future direction of this research.

Bibliography

[1] Bart Baddeley, Paul Graham, Andrew Philippides, and Philip Husbands. Holistic visual encoding of ant-like routes: Navigation without waypoints. *Adaptive Behavior*, 19(1):3–15, 2011.

[2] Ronen Basri, Ehud Rivlin, and Ilan Shimshoni. Visual homing: Surfing on the epipoles. *International Journal of Computer Vision*, 33(2):117–137, 1999.

[3] BA Cartwright and Thomas S Collett. Landmark learning in bees. *Journal of Comparative Physiology*, 151(4):521–543, 1983.

[4] K Cheng, TS Collett, A Pickhard, and R Wehner. The use of visual landmarks by honeybees: Bees weight landmarks according to their distance from the goal. *Journal of Comparative Physiology A*, 161(3):469–475, 1987.

[5] Jonathan Courbon, Youcef Mezouar, and Philippe Martinet. Indoor navigation of a non-holonomic mobile robot using a visual memory. *Autonomous Robots*, 25(3):253–266, 2008.

[6] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[7] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[8] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[9] Douglas D Gaffin, Alexander Dewar, Paul Graham, and Andrew Philippides. Insect-inspired navigation algorithm for an aerial agent using satellite imagery. 2015.

[10] José Gaspar, Niall Winters, and José Santos-Victor. Vision-based navigation and environmental representations with an omnidirectional camera. *Robotics and Automation, IEEE Transactions on*, 16(6):890–898, 2000.

[11] Philippe Gaussier, Cédric Joulain, Stéphane Zrehen, Jean-Paul Banquet, and Arnaud Revel. Visual navigation in an open environment without map. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 545–550. IEEE, 1997.

[12] Suzana Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *Proceedings of the National Academy of Sciences*, 109(Supplement 1):10661–10668, 2012.

[13] Cédric Joulain, Philippe Gaussier, Arnaud Revel, and B Gas. Learning to build visual categories from perception-action associations. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 857–864. IEEE, 1997.

[14] John Lim and Nick Barnes. Robust visual homing with landmark angles. In *Robotics: Science and Systems*. Citeseer, 2009.

[15] Liana M Lorigo, Rodney A Brooks, and WEL Grimsou. Visually-guided obstacle avoidance in unstructured environments. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 1, pages 373–379. IEEE, 1997.

[16] Yoshio Matsumoto, Kazunori Ikeda, Masayuki Inaba, and Hirochika Inoue. Visual navigation using omnidirectional view sequence. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 1, pages 317–322. IEEE, 1999.

[17] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. Visual navigation using view-sequenced route representation. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 83–88. IEEE, 1996.

[18] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. View-based navigation using an omniview sequence in a corridor environment. *Machine vision and applications*, 14(2):121–128, 2003.

[19] Randolf Menzel and Martin Giurfa. Cognitive architecture of a mini-brain: the honeybee. *Trends in cognitive sciences*, 5(2):62–71, 2001.

[20] Michael J Milford and Gordon F Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1643–1649. IEEE, 2012.

[21] Michael Montemerlo and Sebastian Thrun. Fastslam 2.0. *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, pages 63–90, 2007.

[22] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598, 2002.

[23] Hideo Morita, Michael Hild, Jun Miura, and Yoshiaki Shirai. Panoramic view-based navigation in outdoor environments based on support vector learning. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2302–2307. IEEE, 2006.

[24] Kevin P Murphy. Conjugate bayesian analysis of the gaussian distribution. *def*, $1(2\sigma 2)$:16, 2007.

[25] Deon Sabatta and Roland Siegwart. Bearings-only path following with a vision-based potential field. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2755–2760. IEEE, 2014.

[26] J Santos-Victor and Giulio Sandini. Visual-based obstacle detection: a purposive approach using the normal ow. In *Proc. of the International Conference on Intelligent Autonomous Systems, Karlsruhe, Germany*. Citeseer, 1995.

[27] Jiali Shen and Huosheng Hu. Visual navigation of a museum guide robot. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 2, pages 9169–9173. IEEE, 2006.

[28] MV Srinivasan, SW Zhang, JS Chahl, G Stange, and M Garratt. An overview of insect-inspired guidance for application in ground and airborne platforms. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 218(6):375–388, 2004.

[29] Selim Temizer. *Optical flow based local navigation*. PhD thesis, Massachusetts Institute of Technology, 2001.

[30] Rudiger Wehner. Desert ant navigation: how miniature brains solve complex tasks. *Journal of Comparative Physiology A*, 189(8):579–588, 2003.

[31] Rüediger Wehner, Ken Cheng, Holk Cruse, et al. Visual navigation strategies in insects: lessons from desert ants. *New visual neurosciences. MIT Press, Cambridge, MA*, pages 1153–1163, 2014.

[32] Alan M Zhang and Lindsay Kleeman. Robust appearance based visual route following for navigation in large-scale outdoor environments. *The International Journal of Robotics Research*, 28(3):331–356, 2009.

[33] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference (RSS)*, pages 109–111, 2014.

VITA

Alex John Suhren

Candidate for the Degree of

MASTER OF SCIENCE

Thesis: BAYESIAN WEIGHTED SUMMATION OF THE ABSOLUTE DIFFERENCE AND ITS IMPLICATIONS FOR ROBUST VISUAL NAVIGATION

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education: Alex J. Suhren Completed the requirements for a Bachelor of Science in Mechanical Engineering degree Oklahoma State University in December, 2013.

Experience: Vision-based Navigation

Robotic Operating System Programming

Gazebo Simulation Environment

Python Programming